

AD A145 685

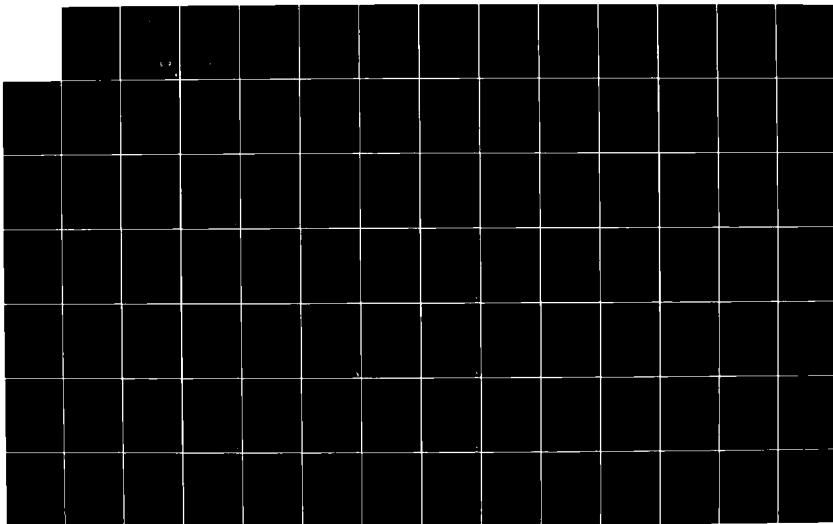
NERF - A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUN. (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARL/STRUC-397

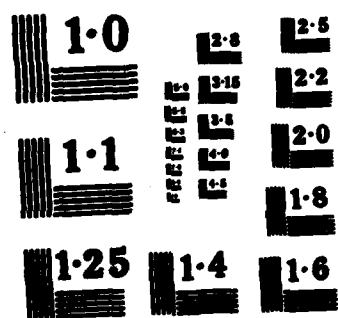
1/ 7

UNCLASSIFIED

F/G 9/2

NI





Copy for NTIS (2)

ARL-STRUC-REPORT-397

AR-002-984



AD-A145 685

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
AERONAUTICAL RESEARCH LABORATORIES
MELBOURNE, VICTORIA

STRUCTURES REPORT 397

NERF - A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUNCTIONS - RELIABILITY MODELLING,
NUMERICAL METHODS AND PROGRAM DOCUMENTATION

by

G.D. MALLINSON AND A.D. GRAHAM

THE UNITED STATES NATIONAL
TECHNICAL INFORMATION SERVICE
IS AUTHORISED TO
REPRODUCE AND SELL THIS REPORT

Approved for Public Release

DTIC FILE COPY

DTIC
ELECTE
SEP 24 1984
S D E

© COMMONWEALTH OF AUSTRALIA 1983

COPY No

SEPTEMBER, 1983

84 07 10 012

AR-002-984

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
AERONAUTICAL RESEARCH LABORATORIES

STRUCTURES REPORT 397

NERF - A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUNCTIONS - RELIABILITY MODELLING,
NUMERICAL METHODS AND PROGRAM DOCUMENTATION

by

G.D. MALLINSON and A.D. GRAHAM

SUMMARY

The computer program NERF (Numerical Evaluation of Reliability Functions) has been designed to evaluate the reliability functions that result from the application of reliability analysis to the fatigue of aircraft structures, in particular, those reliability functions derived by Payne¹⁻⁴ and his co-workers at the Aeronautical Research Laboratories. The NERF program, although based on the Payne reliability models is capable of extension to more complex models as the need arises.

This document details the mathematical development of the reliability functions evaluated by NERF and describes the computer program in sufficient detail to allow desired modifications.



© COMMONWEALTH OF AUSTRALIA 1983

POSTAL ADDRESS: Director, Aeronautical Research Laboratories,
P.O. Box 4331, Melbourne, Victoria, 3001, Australia.

PREFACE

The computer programs described were developed and listed at ARL. The documentation was completed by Dr. Mallinson after he transferred to Auckland University. This work was funded by ARL.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



TABLE OF CONTENTS

1. <u>INTRODUCTION</u>	1
1.1. Report Outline	3
2. <u>RELIABILITY MODELLING</u>	5
2.1. Basic Reliability Functions	7
2.1.1. Definitions	7
2.1.2. Relationships between the basic reliability functions	10
2.2. Derivation of the Most General Functions Evaluated by NERF	11
2.2.1. Definition of random variables	13
2.2.2. Time zones and physical processes	14
2.2.3. Model equations	17
2.2.4. Transformed random variables	21
2.2.5. Expressions for $P_S(t)$	26
2.2.6. The effects of inspections	28
2.2.7. The derivation of risk rates	36
2.2.8. The density function for strength	45
2.2.9. The failure density for strength	47
2.2.10. The probability of failure	51
2.2.11. The probability of detection at an inspection	55
3. <u>DETAILED SPECIFICATIONS FOR THE MODELS EVALUATED BY NERF</u>	59
3.1. Model Classification System	61
3.2. Input Data Specifications	67
3.2.1. Crack growth function	68
3.2.2. Strength decay function	74
3.2.3. Probability of Load Exceedence	80
3.2.4. Inspection removal and crack detection functions	84
3.2.5. Density functions for the basic random variables	88
3.2.6. Model parameters	91
3.2.7. Fatigue life limiting	95

3.3.	Data Limits and Their Effects	97
3.3.1.	Default limits	100
3.3.2.	Notation for limit functions	101
3.3.3.	Conditions for the existence of an integration along the line $R = \alpha \varphi(\xi)$	104
3.3.4.	Conditions for the existence of integrations along the line $\xi = \bar{\xi}_f$	108
3.4.	Specification for the Most General Model	109
3.4.1.	Specifications for the reliability functions as integrations over initial crack length	111
3.4.2.	Specification for the integrand functions for the most general model	115
3.5.	Derivations and Specifications for Simpler Models	118
3.5.1.	Constant relative strength	119
3.5.2.	Constant relative fatigue life	123
3.5.3.	Constant relative strength and relative fatigue life	130
3.6.	The Construction of a Time Sequence	133
3.6.1.	Auxiliary functions	135
3.6.2.	Inspection procedures	138
3.6.3.	Simplifying options	144
4.	<u>NUMERICAL METHODS</u>	146
4.1.	Numerical Integration	148
4.1.1.	Mathematical basis (single integration)	150
4.1.2.	Implementation (single integration)	157
	FUNCTION ADAPT2	168
	SUBROUTINE ADASET	177
	SUBROUTINE ERRORT	179
4.1.3.	Multiple integration - error considerations	183
4.1.4.	Implementation (multiple integration)	191
	FUNCTION ADAPTO	195
	FUNCTION ADAPT1	198
	SUBROUTINE INFINT	199

SUBROUTINE INFLE1	201
SUBROUTINE INFLE2	202
SUBROUTINE INFST	203
SUBROUTINE INFSUP	204
4.1.5. Performance	205
4.2. Interpolation	211
4.2.1. Mathematical basis	212
4.2.2. Implementation	216
4.2.3. Inverse interpolation	218
FUNCTION FINTERP	219
FUNCTION DERIV	222
4.3. Solution of Equations	223
4.3.1. The secant method	224
4.3.2. Implementation	226
FUNCTION FSOLVE	227
FUNCTION FSOLV2	227
4.4. Miscellaneous Support Functions	228
4.4.1. Function range limiting	228
SUBROUTINE RANGE	229
4.4.2. Index location	231
FUNCTION INDHI	232
FUNCTION INDLOW	233
4.4.3. Merging	234
SUBROUTINE MERGE	235
 5. <u>DESCRIPTION OF THE NERF COMPUTER CODE</u>	 237
5.1. General Coding Philosophies and Methods	237
5.2. Principal Phases of Operation and Program Output	240
PROGRAM NERF	241
5.2.1. Data input and initialisation of computational algorithms	243
SUBROUTINE CFIN	246
SUBROUTINE RFSET	251
SUBROUTINE SETTAB	253
5.2.2. Development of a time sequence including inspections	261

SUBROUTINE ADVNCE	262
FUNCTION RSKTOT	274
FUNCTION RSKLOG	279
5.2.3. Calculation of strength distributions	280
SUBROUTINE FLPROB	281
5.2.4. Termination	285
SUBROUTINE CFNEW	286
SUBROUTINE CFOUT	287
5.2.5. Program output	288
SUBROUTINE HEAD	295
SUBROUTINE OUTPUT	296
5.2.6. Graphics operations	297
5.3. The Evaluation of Input Functions	304
5.3.1. Crack growth function	306
FUNCTION CRKDEV	307
FUNCTION CRKGR	308
FUNCTION CRKINV	310
5.3.2. Strength decay function	311
FUNCTION PSI	312
FUNCTION PSIDEV	316
FUNCTION PSINV	317
FUNCTION PSISSET	318
FUNCTION STRFN	319
5.3.3. Risk rates	320
FUNCTION RLOAD	321
FUNCTION RLOSET	322
5.3.4. Inspection removal and crack detection functions	323
FUNCTION SINSF	325
5.3.5. Density functions	327
FUNCTION ALPSET	331
FUNCTION BETCNG	332
FUNCTION BETSET	333
FUNCTION PALPHA	334
FUNCTION PBETA	335
FUNCTION PDF	336
FUNCTION PDFSET	338

	FUNCTION	PRNO	341
	FUNCTION	RNONRM	342
	FUNCTION	RNOSET	343
5.4.	The Evaluation of the Loss Factor		344
	FUNCTION	GVAL	349
5.4.1.	Structure of the interpolation table		350
	FUNCTION	ALPTAB	356
	FUNCTION	INITAB	358
	FUNCTION	RKTAB	360
5.4.2.	Initialisation		362
	SUBROUTINE	NODES	364
	FUNCTION	RINTV	366
	FUNCTION	RLGAM	367
5.4.3.	Interpolation		368
	FUNCTION	GALP	376
	FUNCTION	GSTAR	378
5.4.4.	Special considerations when including virgin risk		382
	SUBROUTINE	GADJST	383
5.5.	The Evaluation of $P_F(t)$, $P_{det}(t)$, $r_s(t)$ and $r_v(t)$		384
5.5.1.	Overview		384
5.5.2.	Probability of failure $P_F(t)$		391
5.5.3.	Risk of static fracture by fatigue		396
5.5.4.	Probability of detection $P_{det}(t)$		399
5.5.5.	Virgin risk and adjustment of $P_S(t)$		404
	FUNCTION	FALP	410
	FUNCTION	FBET	413
	FUNCTION	FDETO	418
	FUNCTION	FPDET	420
	FUNCTION	FRLTO	421
	FUNCTION	FRVO	424
5.6.	Limits and Their Evaluation		425
5.6.1.	Default limits		426
5.6.2.	Limits for a_0 or n_0		430
	SUBROUTINE	ALPHAP	434
	FUNCTION	FNOR	436
	FUNCTION	FRP	437

5.6.3. Limits for β	438
SUBROUTINE BETALM	440
5.6.4. Limits for α	442
5.7. Risk of fatigue life exhaustion	444
FUNCTION FRFO	449
FUNCTION FRF1	452
FUNCTION FRF2	453
5.8. Strength Distributions	454
5.8.1. Probability density for strength	455
FUNCTION FDSR1	460
FUNCTION FLDRO	461
5.8.2. Failure density for strength	464
FUNCTION FLDFO	470
5.9. Basic Communications, Output Procedures and Mathematical Functions	471
5.9.1. Communications	471
SUBROUTINE ECHO	475
FUNCTION INTEST	476
SUBROUTINE INTGIN	477
SUBROUTINE PROMPT	478
SUBROUTINE REALIN	480
SUBROUTINE TXTIN	483
5.9.2. Two-dimensional data array output	484
SUBROUTINE IRROUT	485
SUBROUTINE ARROUT	486
5.9.3. Function file input	487
SUBROUTINE READFN	488
5.9.4. Run time monitoring and program termination	491
SUBROUTINE EXTIME	492
SUBROUTINE FINISH	493
5.9.5. Mathematical functions	494
FUNCTION ALOG1	495
FUNCTION EXP1	496
5.10. Graphics Support	497
5.10.1. Outer integrand plots	499
SUBROUTINE PLTOUT	500
SUBROUTINE PLTSET	503
SUBROUTINE PLTSTR	504

5.10.2. Integrand - function evaluation maps	505
SUBROUTINE INTPLT	512
SUBROUTINE PLTPNT	515
5.10.3. Loss factor maps	517
SUBROUTINE ARRPLT	520
SUBROUTINE RLINE	522
SUBROUTINE GRID	523
SUBROUTINE CONT	524
SUBROUTINE DIAG	526
SUBROUTINE P	527
SUBROUTINE SETPLT	529
5.10.4. Graphics support subroutines	531
SUBROUTINE FSIZE	533
SUBROUTINE PLOTD	535
SUBROUTINE SETGRF	536
SUBROUTINE SCLFCT	541
SUBROUTINE SMOOTH	542
5.10.5. The graphics post processor NERPLT	545
PROGRAM NERPLT	549
SUBROUTINE COMPRS	551
SUBROUTINE FPLOTS	552
SUBROUTINE FPOINT	554
FUNCTION FUNC	555
SUBROUTINE PLTS	556
SUBROUTINE WINDOW	557
5.11. The Data Preparation Program, NERPRE	559
PROGRAM NERPRE	563
SUBROUTINE SELECT	564
5.11.1. Preparation and editing of the control file	565
SUBROUTINE CHANGE	567
SUBROUTINE CHECK	569
SUBROUTINE PDFCNG	570
SUBROUTINE PDFOUT	571
SUBROUTINE VALOUT	572

5.11.2. Preparation of function files	573
SUBROUTINE DATIN	574
SUBROUTINE FCNFIL	575
REFERENCES	577
NOTATION	579
A.1. Cross Reference Listing for Subroutines and Functions	590
A.2. Definitions for Variables in COMMON	595
A.3. Alphabetical Listing of Prompts and Error Messages	604
A.4. Program Assembly	608
DISTRIBUTION	
DOCUMENT CONTROL DATA	

1. INTRODUCTION

The application of reliability analysis to the fatigue of aircraft structures leads to multiple integral expressions for various statistical functions referred to here by the generic term, 'reliability functions'. These functions depend on modelling assumptions regarding the fatigue process and the way that statistical variations or 'randomness' can be accounted for. The particular assumptions and analysis leading to a given set of reliability functions is called a reliability model.

The computer program NERF (Numerical Evaluation of Reliability Functions) has been designed to evaluate the sets of reliability functions for a wide range of models and types of input data. It is the end product of several years of development of numerical techniques and computer codes for the evaluation of reliability functions, in particular those derived by Payne¹⁻⁴ and his co-workers at the Aeronautical Research Laboratories (ARL). Recent theoretical analysis by Mallinson⁵ has established a general technique for the construction of reliability functions from given model assumptions and the relationships between the Payne et al models and other models developed at ARL by Ford⁶⁻⁸ and Hooke⁹⁻¹². The NERF program, although based on the Payne models, now follows the general analysis of Mallinson⁵ and is capable of extension to more complex models as the need arises.

Generally, the detailed reliability functions are complicated and their numerical evaluation requires careful construction of efficient algorithms and subsequent computer code. NERF has been designed to minimise the involvement with this detail required by a user to apply the analysis to a particular set of fatigue data. The program is, in fact, supported by an interactive data preparation program which ensures that a user does not even have to have an extensive knowledge of computer operation. Provision of graphics facilities by NERF and an interactive post-processor further enhance the ease with which the reliability modelling can be applied and the results interpreted.

NERF and these support programs form a complete facility for the application of reliability analysis to the fatigue of aircraft.

The complete documentation for the NERF computer program consists of three reports. The general theoretical basis for the reliability modelling is described by Mallinson⁵. At the other extreme, Mallinson and Graham¹³ detail in a user manual the operation of NERF for the user whose prime interest is the application of the analysis to a particular set of data. This document fills the space in between the other two by detailing the mathematical development of the reliability functions evaluated by NERF and describing the computer program with sufficient detail that a scientist, or computer programmer can understand 'what the program is doing' or make modifications, (probably respectively).

1.1 Report Outline

The document consists of two main sections. The first, comprising Chapters 2 and 3 describes the mathematical development of the reliability functions and provides a numerical specification for the NERF program. The remaining Chapters document the numerical methods and computer programming. Conceivably, (particularly considering the size of the document) these two sections could have written as separate reports. However, it was considered that the heavy dependence of the computer program description on the mathematical section made the single document a preferable alternative.

The organisation of the theoretical section is straightforward. The development of the most general model evaluated by NERF is presented in Chapter 2 and follows the analysis of Mallinson⁵. Chapter 3 expands the analysis of Chapter 2 to include less complex models and the various optional facilities provided by the computer program.

The organisation of the program description is not so straightforward. NERF is a large FORTRAN computer program and consists of over 100 subroutines or function routines. Many of these perform operations or apply numerical analysis techniques which are not particularly related to reliability modelling and can, in fact, be used by other computer programs. The numerical methods for integration, interpolation and the solution of equations are provided by one such group of

subroutines and functions and are described in Chapter 4 prior to the bulk of the program description. This Chapter is organised in the same manner as the remainder of the document in that the description of particular subroutines and functions are included in the Sections and sub-Sections in which the operations they perform are described. An alternative organisation whereby all the subroutines and functions are described in an Appendix to the main report was rejected although particular descriptions could be more readily located in such an organisation, most descriptions would be separated from the the discussion of their operation by considerable volume of text, making reference between the two difficult. A cross reference index is given in an appendix.

Chapter 5 contains the bulk of the computer program description and is organised in a 'top down' manner, from the most general code to the most particular. Section 5.2 describes the overall control code which selects the various operations required to evaluate the functions requested by the user. Sections 5.3 to 5.8 detail the code which evaluates the reliability functions, relying of course on the the numerical methods presented in Chapter 4. Section 5.9 details the basic input, output and terminal communications code and 5.10 describes the subroutines which provide graphics support.

The appendices contain various detailed sets of programming information such as the definitions of variables in COMMON storage, a cross reference guide to prompts and error messages and program assembly.

2. RELIABILITY MODELLING

There are several phenomena (e.g. fatigue, corrosion) which progressively degrade the ability of a structure or component to survive the effects of its environment. Ultimately this degradation process results in the complete failure of the structure at a time which is a function of the environmental and degradation histories. For a given structure, the time of failure will be unique. Unfortunately, the uncertainties in both the environmental history and its effect on the degradation processes make a prior deterministic calculation of the time of failure impossible.

Certain classes of structures, such as aircraft and automobiles, are manufactured in such a way that for a given environment, the degradation processes are similar between members of each class. It is then possible to apply statistical methods to estimate the time of failure based on a knowledge of the mean behaviour of the class, deduced from experiments or from service failures. Similar statistical methods can be used to account for variations in the environment and/or the effect that the environment has on the degradation process.

A reliability model is one such statistical method. The models evaluated by NERF have all been generated by assuming that the mean relationships between strength degradation, crack length and time are known. That between crack length and time is dependent on a time scale which is determined by

the environmental history. For fatigue, this history consists of an applied load sequence and the time scale is determined by the mean load application rate. Variations in structural behaviour are accommodated by introducing random variables as parameters in the strength - crack length - time relationship.

There are several ways in which the random variables can be introduced, leading to several reliability models. For a given application the most appropriate model will be that for which the assumptions behind the introduction of the random variables best suit that application. The availability of adequate data defining the required probability density functions will also influence the choice of model.

The models evaluated by the NERF computer program are similar to those derived by Payne et al¹⁻⁴ and can be derived using the method described by Mallinson⁵. This Chapter describes the application of that method to derive the most general model evaluated by the program. This lays the basis for the detailed model description and specification which follow in Chapter 3.

2.1. Basic Reliability Functions

2.1.1. Definitions

The objective of a reliability model is to evaluate 'reliability functions' which represent the aggregate behaviour of a population of structures which fall within the scope of a given statistical representation. These functions are defined briefly below. More complete definitions are given by Mallinson⁵.

The definitions for these functions are relevant to a population with an infinite number of structures. It is an assumption of the analysis that the number of structures in a population is sufficiently large for the use of continuous functions to be meaningful.

(1) Risk rate $r(t)$

$$r(t) = \text{risk rate} = \frac{\text{Fraction of remaining population failing}}{\text{(unit time), at time } t}. \quad (2.1)$$

In many cases it is convenient to regard the total risk rate as being the sum of several component risks. For the models evaluated by NERF, three components are identified.

$$r(t) = r_s(t) + r_f(t) + r_v(t) \quad (2.2)$$

where $r_s(t)$, $r_f(t)$ and $r_v(t)$ are the risk of static failure by fatigue, the risk of fatigue ^{life exhaustion} and the virgin risk respectively.

Definitions for the component risks are given in section 2.3.

(ii) Probability of failure $P_F(t)$

Let the random variable \underline{F} denote the time of failure of a structure in the population. At time t , the probability of failure is the probability that \underline{F} is less than t . For a large population,

$$\underline{P}_F(t) = \text{Fraction of original population that has failed before time } t, \text{ (including failures at } t). \quad (2.3)$$

(iii) Probability of survival $P_S(t)$

At time t , the probability of survival is the probability that \underline{F} is greater than t . For a large population,

$$\underline{P}_S(t) = \text{Fraction of the original population remaining at time } t. \quad (2.4)$$

Obviously,

$$\underline{P}_S(t) + \underline{P}_F(t) = 1. \quad (2.5)$$

(iv) Probability density of the time to failure $p_F(t)$

The fraction of the population with times of failure in the interval $(t, t + dt)$ is given by $p_F(t)dt$ where $p_F(t)$ is the probability density function for the time to failure.

(v) Density function for strength $p_R(R | F > t)$

At a given time, it can be of considerable interest to know the distribution of strength among the surviving structures. Using R to denote strength, this distribution is given by the conditional density for R given survival to time t , i.e. $p_R(R | F > t)$.

(vi) Failure density for strength $p_R(R | t)$

The conditional density for R given failure at time t is called the failure density for strength.

2.1.2. Relationships between the basic reliability functions

From the definitions of the basic reliability functions, the following relationships can be derived, (Mallinson⁵).

$$p_{\bar{F}}(t) = \frac{dp_{\bar{F}}}{dt} = -\frac{dp_{\bar{S}}}{dt} \quad (2.6)$$

$$r(t) = \frac{dp_{\bar{F}}}{dt} / (1 - p_{\bar{F}}(t)) \quad (2.7)$$

$$= -\frac{dp_{\bar{S}}}{dt} / p_{\bar{S}}(t) \quad (2.8)$$

$$= p_{\bar{F}}(t) / p_{\bar{S}}(t) \quad (2.9)$$

$$p_{\bar{S}}(t) = \exp\left\{-\int_0^t r(t') dt'\right\} \quad (2.10)$$

$$\text{If } p_{\bar{F}}(t) = \int_0^\infty f(R) dR, \text{ then}$$

$$p_{\bar{R}}(R|t) = f(R) / p_{\bar{F}}(t) \quad (2.11)$$

$$\text{If } p_{\bar{S}}(t) = \int_0^\infty g(R) dR, \text{ then}$$

$$p_{\bar{R}}(R|\bar{F}>t) = g(R) / p_{\bar{S}}(t). \quad (2.12)$$

2.2. Derivation of the Most General Functions Evaluated by NERF

The derivation of a reliability model for fatigue relies on the existence of a functional relationship between strength, crack length and time. Parameters in this relationship can account for variations in behaviour between structures in the population, and become the random variables of the reliability model. Given probability density functions for these random variables and a relationship between strength and risk rate, the reliability model can predict the aggregate behaviour of the population in terms of the basic functions defined in the previous section.

The models evaluated by NERF can be derived using the method described by Mallinson⁵ which is summarised by the following steps.

- (i) Random variables representing parameters in the fatigue process are defined.
- (ii) Distinct phases in the fatigue process are identified and each phase associated with an appropriately defined time zone.
- (iii) For each time zone, a relationship between risk rate and the random variables is established via the strength - crack length - time equation.

- (iv) Transformed random variables are defined and the equations established to this stage recast in terms of those variables.
- (v) An integral expression for $P_{\underline{S}}(t)$ is obtained by integrating an expression for the probability of survival for structures with a given set of values of the random variables over the whole space encompassed by those variables (Mallinson⁵).
- (vi) The expression for $P_{\underline{S}}(t)$ is modified to include the effects of inspections.
- (vii) Having generated an expression for $P_{\underline{S}}(t)$, the remaining reliability functions can be obtained using the relations (2.6 - 2.12).

The derivation presented below can be regarded as a particularisation of the more general treatment described by Mallinson⁵ for the models evaluated by NERF.

2.2.1. Definition of random variables

The models depend at most on three random variables.

(i) X_1 : Comparative fatigue life

Let t_f be the fatigue life of a given structure.

Then,

$$x_1 = t_f / \bar{t}_f \quad (2.13)$$

defines X_1 as the comparative fatigue life, where \bar{t}_f is the median fatigue life of the population.

(ii) X_2 : Initial crack length

At the commencement of time, ($t=0$), structures may be cracked. The length of such a crack for a given structure is denoted by X_2 .

(iii) X_3 : Relative residual strength

As a structure ages, its strength decreases as the result of increasing crack length. The ratio of the strength of a given structure at a given age to that of the median structure of the same age is called the relative residual strength X_3 .

It is assumed that suitable density functions can be determined for these random variables.

2.2.2. Time zones and physical processes

Three phases and corresponding time zones are identified. For a given structure the times for transition between zones depend in general on the values of the random variables.

(i) D_1 : Uncracked

Generally, structures commence life uncracked and therefore unweakened. However, this time zone does not exist for a structure which starts life with a pre-existing crack.

(ii) D_2 : Cracked

Following crack initiation the structure weakens as a result of fatigue crack growth. This phase of the fatigue process exists from the initiation time, t_1 , (or 0 for the initially cracked structure) until such time when it is impossible for the structure to sustain even the static loads imposed by its environment. During this time zone the strength reduces from the initial, or virgin strength of the uncracked phase.

(iii) D_3 : Failed

During the first two time zones, structures fail when an applied load exceeds the current strength. Such failures can be expressed as a risk rate as defined below. There are however

certain conditions which are known to lead to structural failure regardless of the applied dynamic loads. These conditions define a third time zone which is given the name 'failed'. It is important to note that any structure that enters D_3 is deemed to have failed but D_3 does not contain all failed structures. The imposition of an upper limit for D_2 is referred to here as fatigue life limiting.

The time of transition from D_2 to D_3 , denoted here by t_f , is a function of the random variables. For a set of structures having the same combination of values of the random variables, this time is the fatigue life of the set as used in equation (2.13); if a structure fails before t_f as the result of an applied load exceeding the current strength, the time of failure for that structure will not equal the fatigue life for the set. This subtle difference between time of failure and fatigue life must be remembered when establishing density functions for \bar{X}_1 .

The models evaluated by NERF can be divided into two groups. The first group assume that all structures commence life cracked, and are called initial crack models to distinguish them from models in the second group in which all structures commence life uncracked. NERF does not model a population in which only some of the structures are initially cracked.

For models in the first group, the time zone D_1 does not exist. For models in the second group, a structure is in D_1 if,

$$t < t_1 \quad . \quad (2.14)$$

where t_1 is, in general, a function of the random variables.

Structures belong to D_2 if

$$t_1 \leq t < t_f \quad (2.15)$$

and D_3 if

$$t > t_f \quad . \quad (2.16)$$

For models in the first group, structures belong to D_2 if

$$0 \leq t < t_f \quad (2.17)$$

and (2.16) defines D_3 .

2.2.3. Model equations

To generate the reliability model, an expression for the risk rate for each time zone in terms of the random variables must be derived. For fatigue, the risk rate is assumed to be the consequence of the applied fluctuating load sequence which has two major effects.

(i) Damage assumption

Each load application produces an increment of damage. If the structure is uncracked the damage produces an inevitable approach to crack initiation. Following initiation, the damage produces an increment in crack length and a subsequent reduction in strength.

(ii) Contribution to risk

Each load application has the potential for producing structural failure if the load exceeds the current strength of the structure.

The damage assumption, and the fact that, in terms of the fatigue life of the structure, the applied loads occur with sufficient frequency that variations in their timing may be ignored, permits a direct link between time and load applications to be made via an average load application rate 1_T . In fact the link between time and load applications was emphasised by Payne et al by the use of the letter N to denote time.

Let R denote the current strength of a structure and L denote load magnitude. From a given load sequence it is possible to regard L as a random variable and derive a stationary probability density $p_L(L)$ for the applied load. The probability distribution of applied loads is,

$$P_L(L) = \int_0^L p_L(L') dL'. \quad (2.18)$$

Because this distribution is stationary, the probability that an applied load exceeds the current strength, R , of a structure is,

$$P(L > R) = 1 - P_L(R) = \bar{P}_L(R). \quad (2.19)$$

The instantaneous risk rate is given by,

$$r_k(t) = \bar{P}_L(R) \cdot 1_r. \quad (2.20)$$

(The subscript, k , is used to denote a time zone. At this stage, equation (2.20) is relevant for any time zone.)

The damage assumption is included by postulating that the strength of the structure is a function of crack length which is, in turn, a continuous function of the number of load applications. Using a to denote crack length,

$$\bar{R} = \bar{R}(a(\tilde{t})) \quad (2.21)$$

which functions are known at least in the mean and have the general form shown in figure 2.1. Functions for particular structures are obtained by introducing parameters into equation (2.21). These parameters become the random variables of the model.

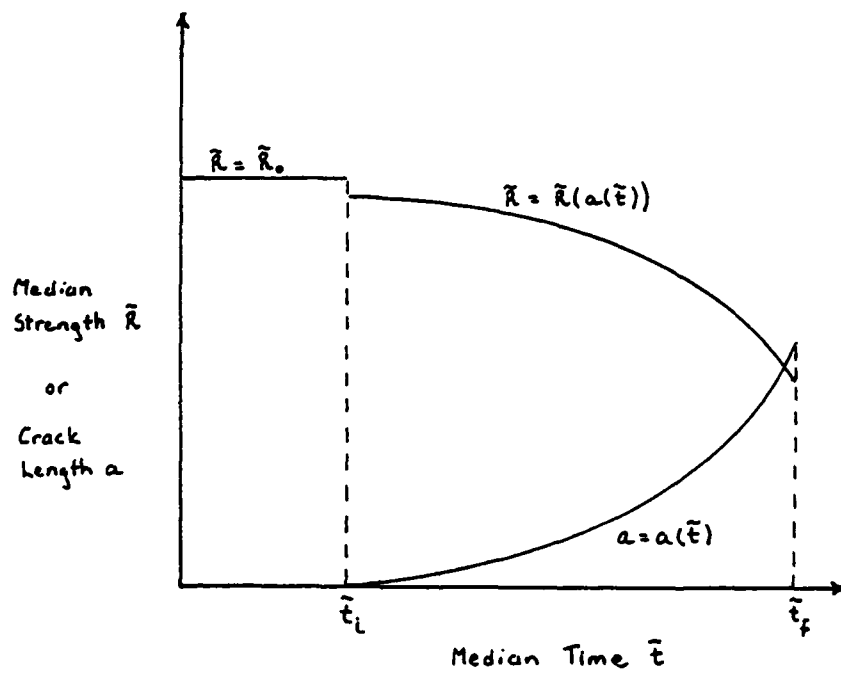


Figure 2.1. General forms for the median strength and crack length functions.

For an uncracked structure in D_1 the strength is given by,

$$R = x_3 \tilde{R}_0 \quad (2.22)$$

where \tilde{R}_0 is the median virgin strength for the population.

For a cracked structure in D_2 the strength is

$$R = x_3 \tilde{R}(a(t/x_1 + a^{-1}(x_2))) \quad (2.23)$$

or

$$R = x_3 \tilde{R}_0 \psi(t/x_1 + \tilde{t}_0) \quad (2.24)$$

where $\psi(\tilde{t})$ is the median relative strength decay function,

$$\psi(\tilde{t}) = \tilde{R}(a(\tilde{t}))/\tilde{R}_0 \quad (2.25)$$

The risk rate for a structure in D_1 is,

$$r_1(x, t) = P_L(x_3 \tilde{R}_0) l_r \quad (2.26)$$

and for one in D_2

$$r_2(x, t) = P_L(x_3 \tilde{R}_0 \psi(t/x_1 + \tilde{t}_0)) \quad (2.27)$$

2.2.4. Transformed random variables

The random variables defined in section 2.2.1. are those in terms of which the reliability models evaluated by NERF were initially defined by Payne et al . Some advantages can be obtained, particularly with respect to the numerical procedures if the models are cast in terms of a new set of transformed random variables. As discussed by Mallinson⁵, the use of transformed random variables facilitates comparison with other reliability models such as those of Ford⁶⁻⁷.

The internal operation of NERF is based on the use of the transformed variables. However input data and computed results are expressed in terms of the original set of variables. It is envisaged that the user of NERF will have little requirement for details of the expressions evaluated by the computer program and the equations presented here are directed to a reader with a requirement to understand exactly how the program works. The reliability functions and the ensuing program specification and description are expressed in terms of the transformed set of random variables.

The notation used here differs from that used by Mallinson⁵ and the conventions used for random variables in the previous sections of this report. This different notation was used during the development of the computer program and it was considered more efficient to retain it in the documentation rather than change the symbolic names throughout the computer code.

The three random variables in the transformed set are defined below.

(i) θ : Age

Significant reductions in the complexity of the reliability functions can be obtained if the argument of ψ is used as a random variable. This argument is a measure of the age of a structure and is related to the original random variables by,

$$\theta = t/\bar{X}_1 + a^{-1}(\bar{X}_2) \quad (2.28)$$

Age is, in fact, equivalent to median time, \bar{t} .

(ii) : Virgin strength

From (2.22), the virgin strength of any structure, α say, is,

$$\alpha = \bar{X}_3 \tilde{R}_0 \quad (2.29)$$

(iii) a_0 : Initial crack length

a_0 is, in fact equivalent to \bar{X}_1 . In some instances it may be more convenient to use n_0 , which from (2.28) is the initial age, i.e.,

$$n_0 = a^{-1}(a_0) \quad (2.30)$$

Using $p_\alpha(\alpha)$, $p_{a_0}(a_0)$ and $p_{n_0}(n_0)$ to denote the density functions for α , a_0 and n_0 respectively,

$$p_\alpha(\alpha) = p_{X_1}(\alpha/\tilde{R}_0)/\tilde{R}_0, \quad (2.31)$$

$$p_{a_0}(a_0) = p_{X_2}(a_0) \quad (2.32)$$

and
$$p_{n_0}(n_0) = p_{X_2}(a(n_0)) \frac{da}{dn_0} \quad (2.33)$$

The density for β is conditional on the value for a_0 or alternatively, n_0 . Using $p_\beta(\beta)$ to denote this conditional density (rather than $p_\beta(\beta|a_0)$ or $p_\beta(\beta|n_0)$,

$$p_\beta(\beta) = p_{X_1}(t/(\beta - n_0)) t/(\beta - n_0)^2. \quad (2.34)$$

In terms of the transformed variables, the risk rate equations are,

$$r_1 = r_1(\alpha) = P_L(\alpha) l_r \quad (2.35)$$

and

$$r_2 = r_2(\alpha, \beta) = P_L(\alpha \gamma(\beta)) l_r. \quad (2.36)$$

The time zones can now be defined as subspaces of the transformed random variable space. Adding to the fatigue life limiting condition the requirement that surviving structures must have strength greater than some minimum value, R_{\min} say, an uncracked structure in D_1 has,

$$0 < \beta < \tilde{t}_1, \quad R_{\min} < \alpha < \infty \quad (2.37)$$

Note that the D_1 subspace exists only when $n_0 \leq \bar{t}_1$, corresponding to $a_0 \leq 0$.

For a cracked structure,

$$\bar{t}_1 < n_0 < \bar{t}_f, \quad n_0 < \beta < \bar{t}_f, \quad R_{\min}/\psi(\beta) < \alpha < \infty. \quad (2.38)$$

Note that the order in which the limits are specified (and hence the order in which the multiple integrations are nested) differs from that used in the theoretical discussion of the genesis of reliability models presented by Mallinson⁵. The computer program always integrates the strength parameter as the innermost integral; the model derivation presented here follows the order most relevant to the computer program.

The time zones are shown as subspaces of the transformed random variables α and β in Fig. 2.2.

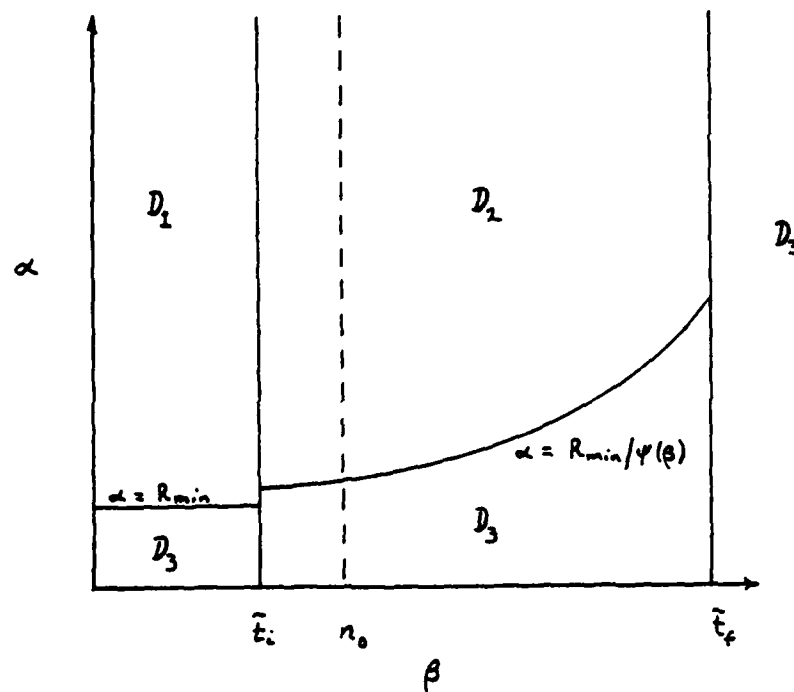


Figure 2.2. Positions of the subspace boundaries in α, β space for constant n_0 . For initial crack models with $n_0 > \tilde{\epsilon}_1$, no structures will exist to the left of the broken line.

- D_1 - Uncracked structures
- D_2 - Cracked structures
- D_3 - Failed structures

2.2.5. Expressions for $P_S(t)$

Given the expressions for the risk rates in each time zone, the probability of survival can be written in the form (Mallinson⁵),

$$P_S(t) = \int_{D_1} H(\underline{x}, t) p_{\underline{X}}(\underline{x}) d\underline{x} + \int_{D_2} H(\underline{x}, t) p_{\underline{X}}(\underline{x}) d\underline{x} \quad (2.39)$$

where $p_{\underline{X}}(\underline{x})$ is the joint density function for the random variables \underline{X} and $H(\underline{x}, t)$ is a loss factor equivalent to the probability of survival for structures with $\underline{X} = \underline{x}$ up to time t .

For $\underline{x} \in D_1$,

$$H(\underline{x}, t) = \exp\{-r_1(\underline{x})t\} \quad (2.40)$$

and for $\underline{x} \in D_2$

$$H(\underline{x}, t) = \exp\{-r_1(\underline{x})t_1 - \int_{t_1}^t r_2(\underline{x}, t') dt'\} . \quad (2.41)$$

In terms of the transformed random variables, the joint density function for α, β and n_0 can be written as

$$\begin{aligned} p_{\alpha, \beta, n_0}(\alpha, \beta, n_0) &= p_{\alpha}(\alpha) p_{n_0}(n_0) p_{\beta}(\beta | n_0) \\ &= p_{\alpha}(\alpha) p_{n_0}(n_0) p_{\beta}(\beta) \end{aligned} \quad (2.42)$$

using the notation of (2.34). For structures in D_1 , with $n_0 = 0$,

$$H(\alpha, \beta, n_0, t) = \exp\{-r_1(\alpha)t\} \quad (2.43)$$

and for structures in D_2 with $n_0 = 0$,

$$H(\alpha, \beta, n_0, t) = \exp\left\{-(t/\beta) \left[r_1(\alpha) \tilde{t}_1 + \int_{\tilde{t}_1}^{\beta} r_2(\alpha, \beta') d\beta' \right] \right\} \quad (2.44)$$

If $n_0 > \tilde{t}_1$, t_1 in (2.41) can be regarded as being zero, D_1 does not exist and in D_2 ,

$$H(\alpha, \beta, n_0, t) = \exp\left\{-\left(t/(\beta - n_0)\right) \int_{n_0}^{\beta} r_2(\alpha, \beta') d\beta'\right\} \quad (2.45)$$

Using the limits defined by equations (2.37-38), the expressions for $P_{\tilde{S}}(t)$ for a population of initially cracked structures is,

$$P_{\tilde{S}}(t) = \int_{\tilde{t}_1}^{\tilde{t}_f} P_{n_0}(n_0) \int_{n_0}^{\tilde{t}_f} P_{\beta}(\beta) \int_{R_{\min}/\psi(\beta)}^{\infty} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta dn_0 \quad \dots (2.46)$$

where $H(\alpha, \beta, n_0, t)$ is defined by equation (2.45).

For a population with $n_0 = 0$, using (2.39),

$$P_{\tilde{S}}(t) = \int_{R_{\min}}^{\infty} p_{\alpha}(\alpha) \exp\{-r_1(\alpha)t\} d\alpha P_f(\tilde{t}_1) + \int_{\tilde{t}_1}^{\tilde{t}_f} P_{\beta}(\beta) \int_{R_{\min}/\psi(\beta)}^{\infty} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \quad (2.47)$$

with the loss factor defined by (2.44).

Equations (2.46) and (2.47) define the probability of survival for the two groups of models evaluated by NERF. All other reliability functions can be derived from these expressions using equations (2.6-2.12).

2.2.6. The effects of inspections

An inspection has two effects on the population of structures.

- (i) Structures that are detected to be deficient according to appropriate criteria are removed.
- (ii) The structures removed during an inspection may be replaced.

The first effect modifies the joint density function for the random variables. Following an inspection at t_{i_j} say, the probability of survival can be evaluated by replacing $p_{\underline{X}}(\underline{x})$ in (2.39) by $p_{\underline{X}}^*(\underline{x})$ where,

$$p_{\underline{X}}^*(\underline{x}) = p_{\underline{X}}(\underline{x})S(\underline{x}, t_{i_j}) \quad (2.48)$$

and $S(\underline{x}, t_{i_j})$ is a function representing the removal of structures during an inspection. The total fraction of the population removed during the inspection is called here the probability of detection at t_{i_j} , $P_{\text{det}}(t_{i_j})$ and is given by,

$$P_{\text{det}}(t_{i_j}) = P_{\underline{S}}^-(t_{i_j}) - P_{\underline{S}}^+(t_{i_j}) \quad (2.49)$$

where $P_{\underline{S}}^-(t_{i_j})$ and $P_{\underline{S}}^+(t_{i_j})$ denote the probability of survival evaluated by equation (2.39) using $p_{\underline{X}}(\underline{x})$ and $p_{\underline{X}}^*(\underline{x})$ respectively.

The second effect is more difficult to model. In general, the replacement structures come from a population with different statistical properties from the original.

Accordingly, the probability of survival following the inspection can be written in the form,

$$P_S(t) = P_S^{j-1}(t) + P_S^j(t-t_{1j}) \cdot P_{det}(t_{1j}) \quad (2.50)$$

where $P_S^{j-1}(t)$ is the probability of survival for the population existing prior to the inspection and $P_S^j(t-t_{1j})$ represents the probability of survival for the population of replacement structures. As the inspections occur, the number of terms in (2.50) increase.

Fortunately, useful calculations can be made without involving the full complexity of a general replacement analysis. NERF offers two options. Replacement structures can be assumed to be repaired in such a way that they are no longer susceptible to a risk of failure (Payne²) or the calculations may proceed without replacement (Hooke⁸). In either case only one population need be considered in the analysis.

Returning to the effect of removal during an inspection, the function $S(x, t_{1j})$ can, in principle, account for failures in the inspection procedure. Earlier versions of the NERF computer program assumed that inspections were perfect and that $S(x, t_{1j})$ was a step function which had the effect of modifying the integration limits in the expressions for $P_S(t)$ and the derived reliability functions.

Two inspection criteria can be applied by NERF.

(1) Crack length

The function S is assumed to have the form,

$$S(x, t_i) = S_j(t_i/x_1 + n_0) H_S(t_d - n_0 - t_i/x_1). \quad (2.51)$$

$H_S(z)$ is the unit step function such that $H_S(z) = 0$ for $z \leq 0$ and $H_S(z) = 1$ for $z > 0$. The mean 'cut off time', \bar{t}_d , corresponds to a crack length a_d beyond which the inspection is known to be perfect, ($\bar{t}_d = a^{-1}(a_d)$). $S_j(x_1, t_i)$ is a removal function the form of which does not change between inspections.

Making the substitution, $x_1 = t/(\beta - n_0)$,

$$S(\alpha, \beta, n_0, t, t_i) = H_S(((\bar{t}_d - n_0)t/t_i + n_0 - \beta)t_i/t) S_j(n_0 + (\beta - n_0)t_i/t). \quad (2.52)$$

For given n_0 , all structures with $\beta \geq n_0 + (\bar{t}_d - n_0)t/t_i$ have been removed by the inspection at t_i . Following a crack length inspection, the limits of integration for cracked structures are,

$$\bar{t}_1 \leq n_0 \leq \bar{t}_d, \quad n_0 \leq \beta \leq \bar{t}_f^*, \quad R_{\min}/\psi(\beta) < \alpha < \infty, \quad (2.53)$$

where

$$\bar{t}_f^* = \min \{ \bar{t}_f, n_0 + (\bar{t}_d - n_0)t/t_i \}, \quad (2.54)$$

and depend on only the last inspection time, t_i . The positions of the D_1 and D_2 subspace boundaries for given n_0 are shown in Figure 2.3.

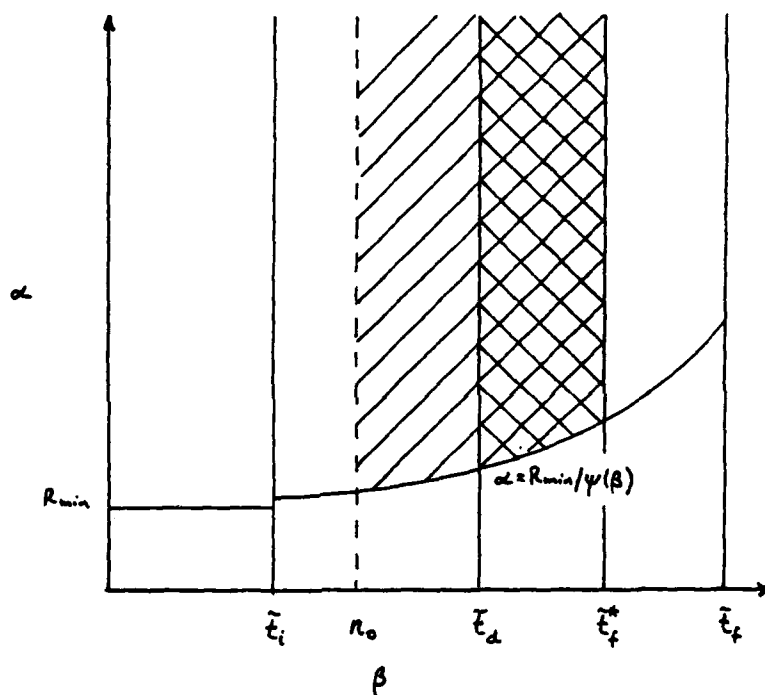
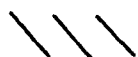


Figure 2.3. Integration domain boundaries in α, β space for constant n_0 following a crack length inspection.



Integration domain for cracked structures.



Potential for rejection by an inspection at the current time.

The function $S_j(n_0 + (\beta - n_0)t_{1j}/t)$ modifies the density function at each inspection so that, following the r 'th inspection,

$$p_\beta^*(\beta) = p_\beta(\beta) \prod_{j=1}^r \{S_j(n_0 + (\beta - n_0)t_{1j}/t)\}. \quad (2.55)$$

Throughout the remainder of this document $p_\beta(\beta)$ will be assumed to be replaced by $p_\beta^*(\beta)$ following inspections.

(ii) Proof load

All structures are subjected to a proof load of magnitude R_p say. All structures with $R \leq R_p$ are rejected so that

$$S(\alpha, \beta, n_0, t, t_{1j}) = H_S(t/(\beta - n_0) - t_{1j}/(\psi^{-1}(R_p/\alpha) - n_0)).$$

..... (2.56)

The domain limits for cracked structures are,

$$\bar{t}_1 \leq n_0 \leq \bar{t}_f, \quad n_0 \leq \beta \leq \bar{t}_f \quad \alpha^* \leq \alpha < \infty \quad (2.57)$$

where

$$\alpha^* = \max \{R_{\min}/\psi(\beta), R_p/\psi(n_0 + (\beta - n_0)t_{1j}/t)\}.$$

.... (2.58)

For uncracked structures, R_p sets a lower limit for α .

The positions of the D_1 and D_2 boundaries for constant n_0 are shown in Figure 2.4.

Note that a proof load test, by its nature, results in a 'perfect' inspection. There is no removal function in this case.

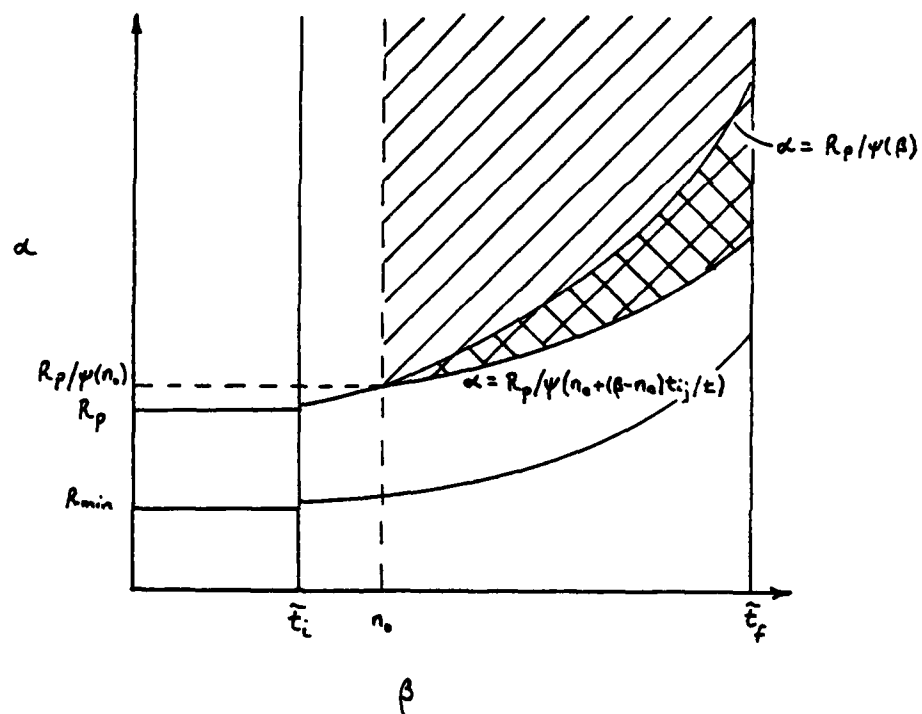
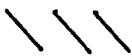


Figure 2.4. Integration domain boundaries following a proof load inspection.



Integration domain for cracked structures.



Potential for rejection by an inspection at the current time.

(iii) Combined

It is possible that both types of test are performed at an inspection following which the more severe of the expressions in (2.53) or (2.57, 2.58) apply. The expression for $\underline{P}_S(t)$ for initially cracked structures is, (c.f. (2.46))

$$\underline{P}_S(t) = \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \int_{n_0}^{\tilde{t}_f^*} p_\beta(\beta) \int_{\alpha}^{\infty} p_\alpha(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta dn_0 \dots \quad (2.59)$$

and for a population without initial cracks, (c.f. (2.47))

$$\underline{P}_S(t) = \int_R^{\infty} p_\alpha(\alpha) \exp\{-r_1(\alpha)t\} d\alpha \bar{P}_\beta(\tilde{t}_1) + \int_{\tilde{t}_1}^{\tilde{t}_f^*} p_\beta(\beta) \int_{\alpha}^{\infty} p_\alpha(\alpha) H(\alpha, \beta, 0, t) d\alpha d\beta \quad (2.60)$$

The subspace boundaries following a combined inspection are shown in figure 2.5.

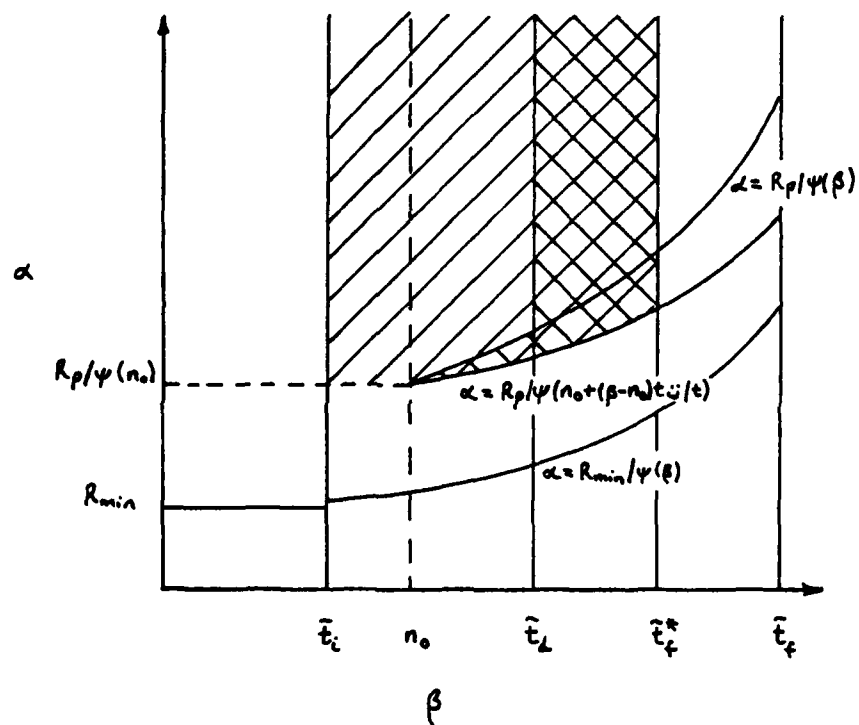


Figure 2.5. Integration domain boundaries following a combined crack length and proof load inspection.



Integration domain for cracked structures.



Potential for rejection by an inspection at the current time.

2.2.7. The derivation of risk rates

The expressions for the total risk rate can be obtained from those for the probability of survival via equation (2.8). The differentiation of expressions involving time dependent transformed random variables requires some care and Mallinson⁵ derived a general result for the case when a single time dependent random variable was the innermost variable of integration. Although this is not the case in equations (2.59) and (2.60), the general result can be applied recursively to yield the required risk rates. An alternative approach would be to rearrange the order of integration so that β was innermost and apply the general result directly. Both approaches yield the same expressions for risk rate.

Taking (2.59) as an example, the subsequent derivation is rendered more compact if the integral expression is written in the form,

$$P_{\underline{S}}(t) = \int_{\bar{t}_1}^{\bar{t}_d} p_{n_0}(n_0) T_{n_0}(n_0) dn_0 \quad (2.61)$$

where,

$$T_{n_0}(n_0) = \int_{n_0}^{\bar{t}_f^*} p_{\beta}(\beta) T_{n_0, \beta}(n_0, \beta) d\beta, \quad (2.62)$$

and

$$T_{n_0, \beta}(n_0, \beta) = \int_{\alpha^*}^{\infty} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha \quad (2.63)$$

Differentiation of equation (2.61) with respect to time yields,

$$P_{\underline{a}}(t) = \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \frac{\partial}{\partial t} T_{n_0}(n_0) dn_0 \quad (2.64)$$

and applying (3.73) obtained by Mallinson⁵,

$$\begin{aligned} \frac{\partial}{\partial t} T_{n_0}(n_0) &= \int_{n_0}^{\tilde{t}_f^*} p_{\beta}(\beta) \left(\frac{\partial}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial}{\partial \beta} \right) T_{n_0, \beta}(n_0, \beta) d\beta \\ &+ \left(\frac{d\tilde{t}_f^*}{dt} - \frac{\partial \beta}{\partial t} \Big|_{\tilde{t}_f^*} \right) T_{n_0, \beta}(n_0, \tilde{t}_f^*) \quad (2.65) \end{aligned}$$

Noting that $\frac{\partial \beta}{\partial t} = (\beta - n_0)/t$; when $\tilde{t}_f^* = n_0 + (\tilde{t}_d - n_0)t/t_{1j}$

$\frac{\partial \beta}{\partial t} \Big|_{\tilde{t}_f^*} = (\tilde{t}_d - n_0)/t_{1j} = \frac{d\tilde{t}_f^*}{dt}$, so that the second term in (2.65) is non-zero only when $\tilde{t}_f^* = \tilde{t}_f$ in which case $\frac{d\tilde{t}_f^*}{dt} = 0$

and $\frac{\partial \beta}{\partial t} \Big|_{\tilde{t}_f^*} = (\tilde{t}_f - n_0)/t$ and (2.65) becomes,

$$\begin{aligned} \frac{\partial}{\partial t} T_{n_0}(n_0) &= \int_{n_0}^{\tilde{t}_f^*} p_{\beta}(\beta) \left(\frac{\partial}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial}{\partial \beta} \right) T_{n_0, \beta}(n_0, \beta) d\beta \\ &- \delta(\tilde{t}_f, \tilde{t}_f^*) \frac{(\tilde{t}_f - n_0)}{t} T_{n_0, \beta}(n_0, \tilde{t}_f) \quad (2.66) \end{aligned}$$

* [Note: $T_{n_0, \beta}(n_0, \tilde{t}_f) = \int_{\mu_f^*}^{\infty} p_{\alpha}(\alpha) H(\alpha, \tilde{t}_f, n_0, t) d\alpha$]

Applying the differential operator in the first term of (2.66) to (2.63) produces,

$$\begin{aligned} & \left(\frac{\partial}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial}{\partial \beta} \right) T_{n_0, \beta}(n_0, \beta) \\ &= \int_{\alpha^*}^{\infty} p_{\alpha}(\alpha) \left(\frac{\partial}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial}{\partial \beta} \right) H(\alpha, \beta, n_0, t) d\alpha \\ & \quad - \left(\frac{\partial \alpha^*}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial \alpha^*}{\partial \beta} \right) p_{\alpha}(\alpha^*) H(\alpha^*, \beta, n_0, t). \end{aligned} \quad (2.67)$$

It can be readily verified that,

$$\begin{aligned} \left(\frac{\partial}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial}{\partial \beta} \right) H(\alpha, \beta, n_0, t) &= \frac{d}{dt} H(\alpha, \beta, n_0, t) \\ &= -r_2(\alpha, \beta, t) H(\alpha, \beta, n_0, t). \end{aligned} \quad (2.68)$$

Substituting (2.67), (2.68) and (2.63) into (2.66)

$$\begin{aligned} \frac{\partial}{\partial t} T_{n_0}(n_0) &= - \int_{n_0}^{\tilde{t}_f^*} p_{\beta}(\beta) \int_{\alpha^*}^{\infty} r_2(\alpha, \beta, t) H(\alpha, \beta, n_0, t) p_{\alpha}(\alpha) d\alpha d\beta \\ & \quad - \int_{n_0}^{\tilde{t}_f^*} \left(\frac{\partial \alpha^*}{\partial t} + \frac{\partial \beta}{\partial t} \frac{\partial \alpha^*}{\partial \beta} \right) p_{\alpha}(\alpha^*) H(\alpha^*, \beta, n_0, t) p_{\beta}(\beta) d\beta \\ & \quad - \delta(\tilde{t}_f, \tilde{t}_f^*) \frac{(\tilde{t}_f - n_0)}{t} \int_{\alpha_f^*}^{\infty} p_{\alpha}(\alpha) H(\alpha, \tilde{t}_f, n_0, t) p_{\beta}(\tilde{t}_f) d\alpha \\ & \quad \dots \dots \dots \end{aligned} \quad (2.69)$$

where

$$\alpha_f^* = \max \{ R_{\min} / \psi(\tilde{t}_f), R_p / \psi(n_0 + (\tilde{t}_f - n_0) t_{1j} / t) \}. \quad (2.70)$$

The second term in (2.69) is non-zero only when $\alpha^* = R_{\min}/\psi(\beta)$, corresponding to those β values for which,

$$R_{\min}/\psi(\beta) \geq R_p/\psi(n_0 + (\xi - n_0)t_{1j}/t). \quad (2.71)$$

Defining $\beta_{p,R_{\min}}$ as the β value at which the equality in (2.71) occurs, it follows from the fact that $R_p > R_{\min}$ that $\beta_{p,R_{\min}} > n_0$.

Moreover, the relationship (2.71) holds only for $\beta \geq \beta_{p,R_{\min}}$. The lower limit of integration for the second term in (2.69) can be replaced by $\beta_{p,R_{\min}}$; for $\beta \geq \beta_{p,R_{\min}}$, $\alpha^* = R_{\min}/\psi(\beta)$ so that $\frac{\partial \alpha^*}{\partial t} = 0$ and the term can be transformed to an integration over α^* , viz.,

$$\begin{aligned} & \int_{\beta_{p,R_{\min}}}^{\tilde{\tau}_f^*} \frac{\partial \beta}{\partial t} \frac{\partial \alpha^*}{\partial \beta} p_{\alpha}(\alpha^*) H(\alpha^*, \beta, n_0, t) p_{\beta}(\beta) d\beta \\ &= \int_{R_{\min}/\psi(\beta_{p,R_{\min}})}^{R_{\min}/\psi(\tilde{\tau}_f^*)} \frac{\partial \beta}{\partial t} \bigg|_{\beta^*} p_{\beta}(\beta^*) p_{\alpha}(\alpha^*) H(\alpha^*, \beta^*, n_0, t) d\alpha^* \\ & \dots \quad (2.72) \end{aligned}$$

where

$$\beta^* = \psi^{-1}(R_{\min}/\alpha^*) \quad (2.73)$$

Note that the term exists only when

$$\psi(\beta_{p,R_{\min}}) > \psi(\tilde{\tau}_f^*) \quad (2.74)$$

Using $\frac{\partial \beta}{\partial t} = (\xi - n_0)/t$ and noting that α^* in (2.72) is now a dummy variable of integration, the second term in (2.69) can be replaced by

$$\int_{R_{\min}/\psi(\beta_{p,R_{\min}})}^{R_{\min}/\psi(\tilde{\tau}_f^*)} \frac{(\xi - n_0)}{t} p_{\alpha}(\alpha) p_{\beta}(\beta^*) H(\alpha, \beta^*, n_0, t) d\alpha.$$

Substitution into (2.61) and applying (2.8) yields,

$$\begin{aligned}
 r(t)P_S(t) &= p_F(t) \\
 &= \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \int_{n_0}^{\tilde{t}_f^*} p_\beta(\beta) \int_{\alpha^*}^{\infty} r_2(\alpha, \beta, t) H(\alpha, \beta, n_0, t) p_\alpha(\alpha) d\alpha d\beta dn_0 \\
 &+ \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \int_{R_{\min}/\gamma(\beta_{P, R_{\min}})}^{R_{\min}/\gamma(\tilde{t}_f^*)} \frac{(\beta^* - n_0)}{t} p_\alpha(\alpha) H(\alpha, \beta^*, n_0, t) p_\beta(\beta^*) d\alpha dn_0 \\
 &+ \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \delta(\tilde{t}_f, \tilde{t}_f^*) p_\beta(\tilde{t}_f) \frac{(\tilde{t}_f - n_0)}{t} \int_{\alpha_f^*}^{\infty} p_\alpha(\alpha) H(\alpha, \tilde{t}_f, n_0, t) d\alpha dn_0 .
 \end{aligned}$$

.... (2.75)

The first term in (2.75) accounts for the failures resulting from loads in the applied fluctuating sequence exceeding the current strengths of the structures in the population and is the contribution referred to here as the risk of static failure by fatigue, $r_s(t)$. The second term accounts for failures resulting from the strength being reduced to R_{\min} and the third term represents structures that have reached their fatigue life limit. Both terms combine to yield the risk of fatigue life exhaustion, $r_f(t)$.

The expression (2.75) is incomplete in that generally the integrands in the last two terms are zero for some values of n_0 . To derive explicit integration limits it is necessary to consider the limiting processes implied by the inspection procedures. The general situation for given n_0 and time t greater than ti_j is shown in figure 2.6. The second term in (2.75), being an integration along the line $R_{\min} = \alpha\psi(\beta)$, will exist only if $\beta_{p,R_{\min}} < \tilde{t}_f^*$. This condition is met if,

$$R_{\min}/\psi(\tilde{t}_f^*) > R_p/\psi(n_0 + (\tilde{t}_f^* - n_0)ti_j/t) \quad (2.76)$$

(The left hand side of (2.76) is the α value of the intersection of the minimum strength boundary with $\beta = \tilde{t}_f^*$; the right hand side is the α value of the intersection of the proof load boundary with $\beta = \tilde{t}_f^*$.)

For $\tilde{t}_f^* = n_0 + (\tilde{t}_d - n_0)t/ti_j$, equation (2.76) can be reduced in the following manner.

$$\begin{aligned} (2.76) &\Rightarrow R_{\min}/\psi(\tilde{t}_f^*) > R_p/\psi(\tilde{t}_d) \\ &\Rightarrow \tilde{t}_f^* > \psi^{-1}(R_{\min}\psi(\tilde{t}_d)/R_p), \text{ (assuming } \psi \text{ is monotonic} \\ &\quad \text{and decreasing),} \\ &\Rightarrow n_0 < (\tilde{t}_d t - \psi^{-1}(R_{\min}\psi(\tilde{t}_d)/R_p)ti_j)/(t - ti_j). \end{aligned} \quad (2.77)$$

For $\tilde{t}_f^* = \tilde{t}_f$,

$$\begin{aligned} (2.76) &\Rightarrow R_{\min}/\psi(\tilde{t}_f) > R_p/\psi(n_0 + (\tilde{t}_f - n_0)ti_j/t) \\ &\Rightarrow n_0 < (t\psi^{-1}(R_p\psi(\tilde{t}_f)/R_{\min}) - \tilde{t}_f ti_j)/(t - ti_j). \end{aligned} \quad (2.78)$$

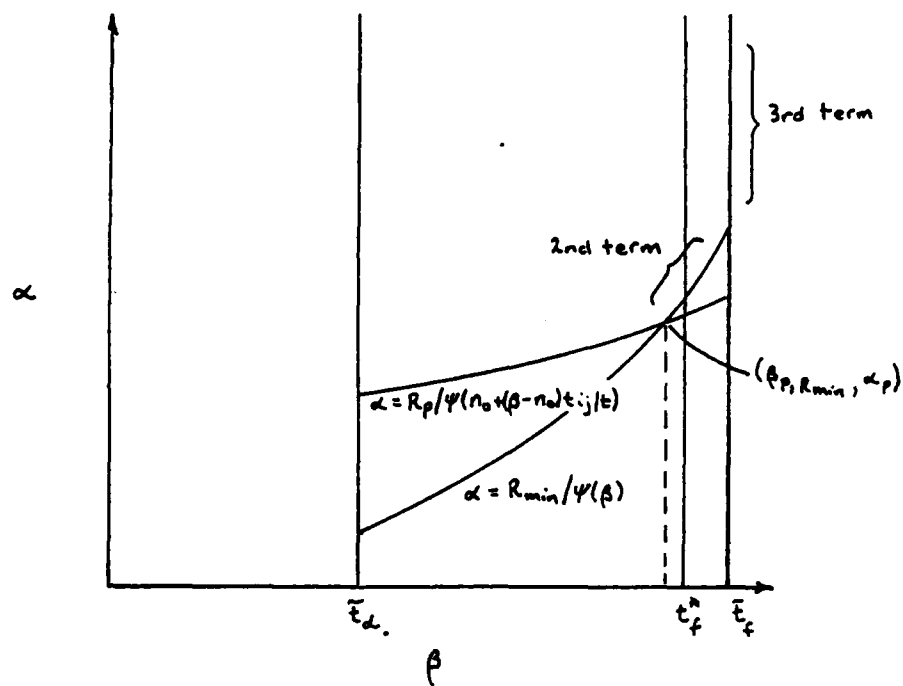


Figure 2.6. Schematic representation of the origins of the second and third terms in equation (2.75).

Both conditions can be combined to yield,

$$\begin{aligned}
 n_0 &< (\min\{\tilde{t}_d, \psi^{-1}(R_p \psi(\tilde{t}_f)/R_{\min})\} t \\
 &\quad - \min\{\tilde{t}_f, \psi^{-1}(R_{\min} \psi(\tilde{t}_d)/R_p)\} t_{l_j}) / (t - t_{l_j}) \\
 &< f_{n_0}(R_{\min}), \text{ say.}
 \end{aligned}
 \tag{2.79}$$

The third term in (2.75) is an integration along the line $\beta = \tilde{t}_f$ and exists only when $\tilde{t}_f^* \geq \tilde{t}_f$. This condition is equivalent to,

$$\begin{aligned}
 n_0 &< (\tilde{t}_d t - \tilde{t}_f t_{l_j}) / (t - t_{l_j}) \\
 &< f_{n_0}, \text{ say.}
 \end{aligned}
 \tag{2.80}$$

Equations (2.79) and (2.80) define the upper limits for the n_0 integration in the second and third (or r_f) terms in equation (2.75).

In a similar manner to that described above for (2.59), equation (2.60) can be differentiated to derive the risk rate for a population without initial cracks.

The result of this differentiation is,

$$\begin{aligned}
r(t)P_{\underline{S}}(t) = & \int_{R_p}^{\infty} p_{\alpha}(\alpha) r_1(\alpha) \exp\{-r_1(\alpha)t\} d\alpha \cdot P_{\beta}(\tilde{t}_1) \\
& + \int_{\tilde{t}_1}^{\tilde{t}_f^*} p_{\beta}(\beta) \int_{\alpha^*}^{\infty} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) r_2(\alpha, \beta) d\alpha d\beta \\
& + \int_{R_{\min}/\psi(\beta_{p, R_{\min}})}^{R_{\min}/\psi(\tilde{t}_f^*)} \frac{\beta^*}{t} p_{\alpha}(\alpha) H(\alpha, \beta^*, n_0, t) p_{\beta}(\beta^*) d\alpha \\
& + \delta(\tilde{t}_f, \tilde{t}_f^*) p_{\beta}(\tilde{t}_f) \frac{\tilde{t}_f}{t} \int_{\alpha_f^*}^{\infty} p_{\alpha}(\alpha) H(\alpha, \tilde{t}_f, n_0, t) d\alpha. \quad (2.81)
\end{aligned}$$

The last three terms of (2.81) are obvious simplifications of (2.75) with $n_0 = 0$. The first term accounts for the failure of uncracked structures in D_1 and contributes to the virgin risk.

2.2.8. The density function for strength

Given an expression for the probability of survival, the density function for strength can be obtained using equation (2.12). Using the relationship,

$$R = \alpha \gamma(\beta) \quad , \quad (2.82)$$

the integration over α in (2.59) can be transformed to one over R , so that,

$$P_S(t) = \int_{\tilde{t}_1}^{\tilde{t}_d} P_{n_0}(n_0) \int_{n_0}^{\tilde{t}_f^*} P_\beta(\beta) \int_{R^*}^{\infty} P_\alpha(R/\gamma(\beta)) \frac{H(R/\gamma(\beta), \beta, n_0, t) dR d\beta dn_0}{\gamma(\beta)} \quad (2.83)$$

.....

where

$$R^* = \alpha^* \gamma(\beta) = \max\{R_{\min}, R_p \gamma(\beta) / \gamma(n_0 + (\beta - n_0) t_{1j} / t)\} \quad (2.84)$$

Rearranging the order of integration yields,

$$P_S(t) = \int_{R_{\min}}^{\infty} \int_{\tilde{t}_1}^{f_{n_0}(R)} P_{n_0}(n_0) \int_{\max(\beta_{p,R}, n_0)}^{\tilde{t}_f^*} \frac{P_\beta(\beta) P_\alpha(R/\gamma(\beta)) H(R/\gamma(\beta), \beta, n_0, t) d\beta dn_0 dR}{\gamma(\beta)} \quad (2.84a)$$

.....

where $\beta_{p,R}$ is the β value for which,

$$R = R_p \gamma(\beta) / \gamma(n_0 + (\beta - n_0) t_{1j} / t) \quad (2.85)$$

and $f_{n_0}(R)$ is defined by (2.79), with R_{\min} replaced by R .

Using (2.12),

$$\begin{aligned}
 & p_R(R | \bar{F} > t) \\
 &= \frac{1}{P_S(t)} \int_{\tilde{t}_1}^{n_0(R)} p_{n_0}(n_0) \int_{\max\{\beta_p, R\}}^{\tilde{t}_f^*} \frac{p_\beta(\beta) p_\alpha(R/\psi(\beta)) H(R/\psi(\beta), \beta, n_0, t) d\beta dn_0}{\psi(\beta)} \\
 & \dots \dots \dots (2.86)
 \end{aligned}$$

Starting with (2.60), the probability of survival for a population without initial cracks can be written in the form,

$$\begin{aligned}
 P_S(t) &= \int_{R_p}^{\infty} p_\alpha(R) \exp\{-r_1(R)t\} dR \cdot \bar{P}_\beta(\tilde{t}_1) \\
 &+ \int_{R_{\min}}^{\infty} \int_{\max\{\tilde{t}_1, \beta_p, R\}}^{\tilde{t}_f^*} \frac{p_\beta(\beta) p_\alpha(R/\psi(\beta)) H(R/\psi(\beta), \beta, 0, t) d\beta dR}{\psi(\beta)} \\
 & \dots \dots \dots (2.87)
 \end{aligned}$$

and the density function for strength becomes,

$$\begin{aligned}
 p_R(R | \bar{F} > t) &= \left[H_s(R - R_p) p_\alpha(R) \exp\{-r_1(R)t\} p_\beta(\tilde{t}_1) + \right. \\
 & \left. \int_{\max\{\tilde{t}_1, \beta_p, R\}}^{\tilde{t}_f^*} \frac{p_\beta(\beta) p_\alpha(R/\psi(\beta)) H(R/\psi(\beta), \beta, 0, t) d\beta}{\psi(\beta)} \right] / P_S(t) \\
 & \dots \dots \dots (2.88)
 \end{aligned}$$

2.2.9. The failure density for strength

Given an expression for $p_F(t)$, the failure density for strength can be obtained using equation (2.11). Starting with (2.75), the α integration can be transformed where possible to an R integration,

$$\begin{aligned}
 p_F(t) = & \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \int_{n_0}^{\tilde{t}_f^*} p_\beta(\beta) \int_{R^*}^{\infty} r_2(R/\psi(\beta), \beta) p_\alpha(R/\psi(\beta)) \chi \\
 & \times \frac{H(R/\psi(\beta), \beta, n_0, t) dR d\beta dn_0}{\psi(\beta)} \\
 & + \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \int_{R_{\min}/\psi(\beta, R_{\min})}^{R_{\min}/\psi(\tilde{t}_f^*)} (\beta^* - n_0) R_\alpha(\alpha) \frac{H(\alpha, \beta^*, n_0, t) p_\beta(\beta^*) d\beta dn_0}{t} \\
 & + \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) p_\beta(\tilde{t}_f) \frac{(\tilde{t}_f - n_0)}{t} \int_{R_f^*(n_0)}^{\infty} \frac{p_\alpha(R/\psi(\tilde{t}_f)) H(R/\psi(\tilde{t}_f), \tilde{t}_f, n_0, t) dR dn_0}{\psi(\tilde{t}_f)} \\
 & \dots \quad (2.89)
 \end{aligned}$$

where

$$R_f^*(n_0) = \max\{R_{\min}, R_p \psi(\tilde{t}_f) / \psi(n_0 + (\tilde{t}_f - n_0) t t_j / t)\}. \quad (2.90)$$

The second term cannot be transformed since each inner integration is along the line $R = R_{\min}$.

Rearranging the order of integration yields,

$$\begin{aligned}
 p_f(t) = & \int_{R_{\min}}^{\infty} r_2(R) \int_{\tilde{t}_1}^{f_{n_0}(R)} p_{n_0}(n_0) \int_{\max\{n_0, \beta_{p,R}\}}^{\tilde{t}_f^*} \frac{p_\beta(\beta) p_\alpha(R/\psi(\beta)) H(R/\psi(\beta), \beta, n_0, t) d\beta dn_0 dR}{\psi(\beta)} \\
 & + \int_{\tilde{t}_1}^{f_{n_0}(R_{\min})} p_{n_0}(n_0) \int_{R_{\min}/\psi(\tilde{t}_f^*)}^{R_{\min}/\psi(\tilde{t}_f^*)} (\beta^* - n_0) \frac{p_\alpha(\alpha) p_\beta(\beta^*) H(\alpha, \beta^*, n_0, t) d\alpha dn_0}{t} \\
 & + \int_{R_f^*(\tilde{t}_1)}^{\infty} \int_{\tilde{t}_1}^{f_{n_0}^d(R)} p_{n_0}(n_0) p_\beta(\tilde{t}_f) \frac{(\tilde{t}_f - n_0) p_\alpha(R/\psi(\tilde{t}_f)) H(R/\psi(\tilde{t}_f), \tilde{t}_f, n_0, t) dn_0 dR}{\psi(\tilde{t}_f)} \\
 & \dots \dots \quad (2.91)
 \end{aligned}$$

where

$$f_{n_0}^d(R) = (\psi^{-1}(R_p \psi(\tilde{t}_f)/R) t - \tilde{t}_f t_{1j}) / (t - t_{1j}) \quad (2.92)$$

(Note that $f_{n_0}^d(R)$ is equivalent to f_{n_0} as defined by (2.80) but with \tilde{t}_d replaced by $\psi^{-1}(R_p \psi(\tilde{t}_f)/R)$.)

Using (2.11), the failure density for strength is,

$$\begin{aligned}
 p_R(R|t) = & \frac{r_2(R)}{p_f(t)} \int_{\tilde{t}_1}^{f_{n_0}(R)} p_{n_0}(n_0) \int_{\max\{n_0, \beta_{p,R}\}}^{\tilde{t}_f^*} \frac{p_\beta(\beta) p_\alpha(R/\psi(\beta)) H(R/\psi(\beta), \beta, n_0, t) d\beta dn_0}{\psi(\beta)} \\
 & + \frac{\delta(R, R_{\min})}{t \cdot p_f(t)} \int_{\tilde{t}_1}^{f_{n_0}(R_{\min})} p_{n_0}(n_0) \int_{R_{\min}/\psi(\beta_{p,R_{\min}})}^{R_{\min}/\psi(\tilde{t}_f^*)} (\beta^* - n_0) p_\alpha(\alpha) p_\beta(\beta^*) H(\alpha, \beta^*, n_0, t) d\alpha dn_0 \\
 & + \frac{H(R - R_f^*(\tilde{t}_1)) p_\beta(\tilde{t}_f)}{t \cdot p_f(t) \psi(\tilde{t}_f)} \int_{\tilde{t}_1}^{f_{n_0}^d(R)} p_{n_0}(n_0) p_\alpha(R/\psi(\tilde{t}_f)) (\tilde{t}_f - n_0) H(R/\psi(\tilde{t}_f), \tilde{t}_f, n_0, t) dn_0 \\
 & \dots \dots \quad (2.93)
 \end{aligned}$$

Note that the second term in (2.93) represents a concentrated density for failures with $R = R_{\min}$. The third term exists only when $R > R_f^*(\tilde{t}_1)$, i.e.,

$$R > \max\{R_{\min}, R_p(\tilde{t}_f)/\psi(\tilde{t}_1 + (\tilde{t}_f - \tilde{t}_1)t_{1j}/t)\}. \quad (2.94)$$

For a population without initial cracks, the α integrations in (2.8f) can be transformed and the order of integration rearranged so that,

$$\begin{aligned} P_F(t) = & \int_{R_p}^{\infty} p_{\alpha}(R) r_1(R) \exp\{-r_1(R)t\} dR \cdot P_{\theta}(\tilde{t}_1) \\ & + \int_{R_{\min}}^{\infty} r_2(R) \int_{\max\{\tilde{t}_1, \beta_{p,R}\}}^{\tilde{t}_f^*} p_{\theta}(\beta) p_{\alpha}(R/\psi(\beta)) \frac{H(R/\psi(\beta), \beta, 0, t)}{\psi(\beta)} d\beta dR \\ & + \int_{R_{\min}/\psi(\beta_{p,R_{\min}})}^{R_{\min}/\psi(\tilde{t}_f^*)} \frac{\beta^*}{t} p_{\alpha}(\alpha) H(\alpha, \beta^*, 0, t) p_{\theta}(\beta^*) d\alpha \\ & + \int_{R_f^*(0)}^{\infty} p_{\theta}(\tilde{t}_f) \tilde{t}_f p_{\alpha}(R/\psi(\tilde{t}_f)) \frac{H(R/\psi(\tilde{t}_f), \tilde{t}_f, 0, t)}{\psi(\tilde{t}_f)t} dR \end{aligned} \quad (2.95)$$

where $R_f^*(0)$ is given by (2.90) with $n_0 = 0$, i.e.,

$$R_f^*(0) = \max\{R_{\min}, R_p \psi(\tilde{t}_f)/\psi(\tilde{t}_f t_{1j}/t)\}. \quad (2.96)$$

It follows that

$$\begin{aligned}
 p_R(R|t) = & \frac{H_s(R-R_p)p_\alpha(R)r_1(R)\exp\{-r_1(R)t\}\bar{p}_\beta(\tilde{t}_1)}{p_F(t)} \\
 & + \frac{r_2(R)}{p_F(t)} \int_{\max\{\tilde{t}_1, \beta_{p,R}\}}^{\tilde{t}_1^*} \frac{p_\beta(\beta)p_\alpha(R/\psi(\beta))H(R/\psi(\beta), \beta, 0, t)}{\psi(\beta)} d\beta \\
 & + \frac{\delta(R, R_{\min})}{p_F(t) \cdot t} \int_{R_{\min}/\psi(\beta_{p,R_{\min}})}^{R_{\min}/\beta(\tilde{t}_1^*)} \beta^* p_\alpha(\alpha) H(\alpha, \beta^*, 0, t) p_\beta(\beta^*) d\alpha \\
 & + \frac{H_s(R-R_F^*(0))p_\beta(\tilde{t}_F)\tilde{t}_F p_\alpha(R/\psi(\tilde{t}_F))H(R/\psi(\tilde{t}_F), \tilde{t}_F, 0, t)}{\psi(\tilde{t}_F)p_F(t) \cdot t} \\
 & \dots\dots (2.97)
 \end{aligned}$$

2.2.10. The probability of failure

Equation (2.5) provides a trivial means of computing $P_F(t)$ given a value for $P_S(t)$ and there appears to be no need to derive an integral expression for the probability of failure.

However, the numerical evaluation of expressions such as (2.59) and (2.60) is difficult when $P_S(t) \approx 1$. To obtain meaningful estimates of $P_S(t)$ near unity the numerical integration must be computed to an accuracy of several significant digits. A better approach in such cases is to evaluate the integral expression for $P_F(t)$ and use (2.5) to obtain $P_S(t)$.

In the practical application of reliability analysis to fatigue, it is normal to concentrate interest on situations where the risk rates are very small and $P_S(t) \approx 1$. Thus, although the integral expression for $P_S(t)$ is fundamental for the derivation of expressions for other reliability functions, it is not used directly in NERF. Instead, the expression for $P_F(t)$ is evaluated. It assumed that NERF will not be applied to situations where the probability of survival is very small (less than 0.01 say) and no provision has been made for this case.

The integral expression for $P_F(t)$ corresponding to (2.39) is (see Mallinson⁵),

$$P_F(t) = \int_{D_1} (1 - H(\underline{x}, t)) p_{\underline{x}}(\underline{x}) d\underline{x} + \int_{D_2} (1 - H(\underline{x}, t)) p_{\underline{x}}(\underline{x}) d\underline{x} + \int_{D_3} p_{\underline{x}}(\underline{x}) d\underline{x} + \dots \quad (2.98)$$

The third term in (2.98) is an integration over the subspace D_3 which contains all those structures that have satisfied a failure criterion (such as minimum strength or finite fatigue life) or have been rejected by a previous inspection.

Referring to Figure 2.7, the third term in (2.98) for initially cracked structures is represented by three terms; the first two terms cover the regions,

$$n_0 \leq \beta < \bar{t}_f^*, \quad -\infty < \alpha < \alpha^* \quad (2.99)$$

and

$$\bar{t}_f^* \leq \beta < \infty \quad -\infty < \alpha < \infty, \quad (2.100)$$

respectively for $n_0 < \bar{t}_d$. The third term accounts for all structures for which $\bar{t}_d \leq n_0$. The expression for $P_F(t)$ is,

$$\begin{aligned} P_F(t) = & \int_{\bar{t}_1}^{\bar{t}_d} P_{n_0}(n_0) \int_{n_0}^{\bar{t}_f^*} P_\beta(\beta) \int_{\alpha^*}^{\infty} (1 - H(\alpha, \beta, n_0, t)) P_\alpha(\alpha) d\alpha d\beta dn_0 \\ & + \int_{\bar{t}_1}^{\bar{t}_d} P_{n_0}(n_0) \left[\int_{n_0}^{\bar{t}_f^*} P_\beta(\beta) \int_{-\infty}^{\alpha^*} P_\alpha(\alpha) d\alpha + \bar{P}(\bar{t}_f^*) \right] dn_0 \\ & + \bar{P}_{n_0}(\bar{t}_d) \end{aligned} \quad (2.101)$$

(The second term in (2.101) includes the two terms described by (2.99) and (2.100))

Equation (2.101) applies for time following an inspection. All structures removed during inspections are included in the integral expression for $P_F(t)$ so that the resulting estimate for $P_F(t)$ does not include any replacement structures.

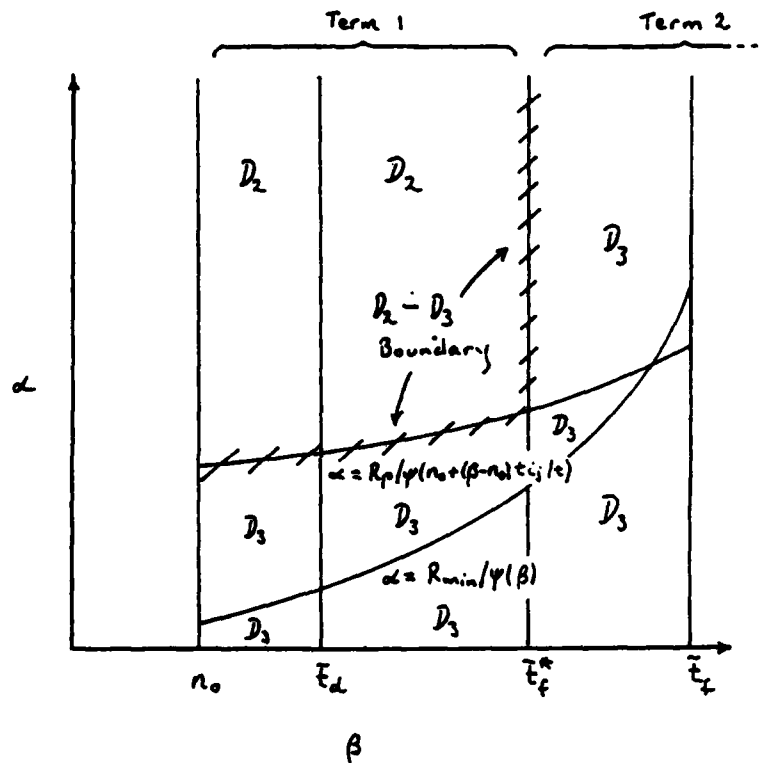


Figure 2.7. Schematic representation of the D_2/D_3 boundary for given n_0 when D_3 includes failed structures together with those rejected by previous inspections.

Also indicated are the regions covered by the first two terms representing D_3 in $P_p(t)$, (equations 2.99 and 2.100 define the limits).

For a population without initial cracking the expression for $P_{\bar{P}}(t)$ is,

$$\begin{aligned}
 P_{\bar{P}}(t) = & \int_{\bar{P}}^{\infty} p_{\alpha}(\alpha) [1 - \exp\{-r_1(\alpha)t\}] d\alpha \cdot \bar{P}_{\beta}(\tilde{t}_1) + \int_{-\infty}^R p_{\alpha}(\alpha) d\alpha \\
 & + \int_{\tilde{t}_1}^{\tilde{t}_f^*} p_{\beta}(\beta) \int_{\alpha}^{\infty} (1 - H(\alpha, \beta, 0, t)) p_{\alpha}(\alpha) d\alpha d\beta \\
 & + \int_{\tilde{t}_1}^{\tilde{t}_f^*} p_{\beta}(\beta) \int_{-\infty}^{\alpha} p_{\alpha}(\alpha) d\alpha + \bar{P}_{\beta}(\tilde{t}_f^*) \quad (2.102)
 \end{aligned}$$

2.2.11. The probability of detection at an inspection

The total fraction of the population removed during an inspection is called the probability of detection and is defined by equation (2.49). Normally, the integral expressions for $P_S(t)$ or $P_F(t)$ evaluated before and after the inspection can be differenced to yield the probability of detection. However, NERF offers a simplifying approximation (see Section 3.6.2) whereby the probability of survival is computed via equation (2.10) from the total risk rate. For this case, an integral expression for the probability of detection must be available.

To derive a suitable expression, recall equation (2.49), viz,

$$P_{\text{det}}(t_{i_{j+1}}) = P_S^-(t) - P_S^+(t),$$

where $P_S^-(t)$ is simply the probability of survival at $t = t_{i_{j+1}}$, with the limits of integration being dependent on the last inspection time, t_{i_j} . $P_S^-(t)$ is obtained by replacing t_{i_j} in the limit equations by t , the current inspection time, so that,

$$\text{(from (2.54))} \quad \tilde{t}_f^{*+} = \min\{\tilde{t}_f, \tilde{t}_d\} \quad (2.103)$$

$$\text{and (from (2.57) with } t_{i_j} \text{ by } t), \quad \alpha^{*+} = \max\{R_{\min}/\psi(\beta), R_p/\psi(\beta)\}. \quad (2.104)$$

The expression for the probability of detection is,

$$P_{\text{det}}(t_{i_{j+1}}) = \int_{\tilde{t}_1}^{\tilde{t}_d} P_{N_0}(n_0) \left[\int_{n_0}^{\tilde{t}_f^{*+}} p_\beta(\beta) \int_{\alpha}^{\infty} p_\alpha(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \right. \\ \left. - \int_{n_0}^{\min\{\tilde{t}_f, \tilde{t}_d\}} p_\beta(\beta) \int_{\max\{R_{\min}/\psi(\beta), R_p/\psi(\beta)\}}^{\infty} p_\alpha(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \right] dn_0 \quad \dots (2.105)$$

The term in square brackets in (2.105) covers the area shown in Figure 2.8. The integration can be rearranged to yield,

$$P_{det}(t_{j+1}) = \int_{\tilde{t}_1}^{\tilde{t}_d} p_{n_0}(n_0) \left[\int_{n_0}^{\tilde{t}_d} p_{\beta}(\beta) \int_{\alpha^*}^{R_p/\psi(\beta)} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta + \int_{\tilde{t}_d}^{\tilde{t}_1^*} p_{\beta}(\beta) \int_{\alpha^*}^{\infty} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \right] dn_0. \quad (2.106)$$

Similarly for a population without initial cracking,

$$P_{det}(t_{j+1}) = \int_{\tilde{t}_1}^{\tilde{t}_d} p_{\beta}(\beta) \int_{\alpha^*}^{R_p/\psi(\beta)} p_{\alpha}(\alpha) H(\alpha, \beta, 0, t) d\alpha d\beta + \int_{\tilde{t}_d}^{\tilde{t}_1^*} p_{\beta}(\beta) \int_{\alpha^*}^{\infty} H(\alpha, \beta, 0, t) p_{\alpha}(\alpha) d\alpha d\beta. \quad (2.107)$$

Note that there is no contribution to (2.107) arising from uncracked structures. An inspection based on a crack length criterion will not reject uncracked structures. A proof load test will reject uncracked structures at the first inspection only. For this inspection, the term,

$$P_{\beta}(\tilde{t}_1) \int_{R_{min}}^{R_p} p_{\alpha}(\alpha) \exp\{-r_1(\alpha)t\} d\alpha \text{ must be added to (2.107)}$$

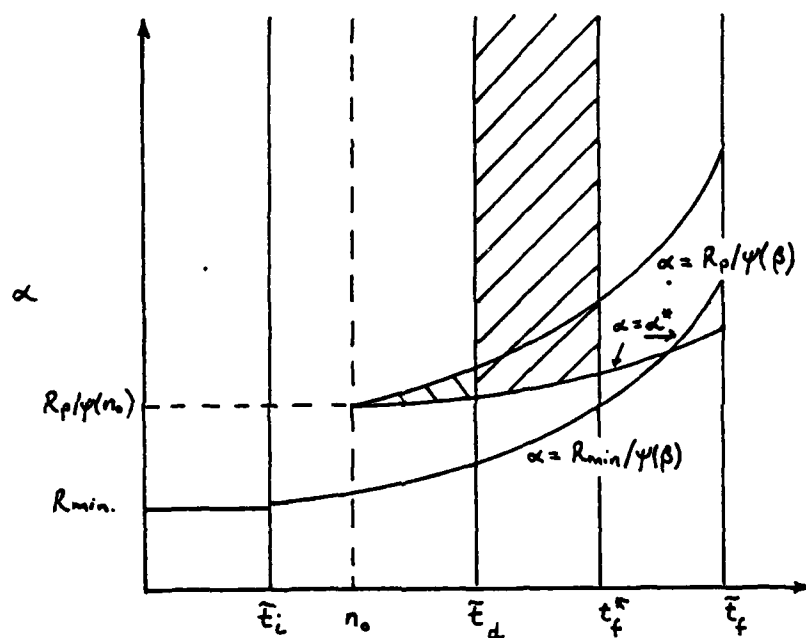


Figure 2.8. Area in α, β space, for given n_0 , representing the probability of detection.

The area can be divided into two terms.

$$\left. \begin{array}{l} \backslash \backslash \backslash \text{ term 1} \\ // // // \text{ term 2} \end{array} \right\} \text{ (equation 2.106).}$$

Note that this argument used to derive the probability of detection ignores the effect of the removal function for a crack length inspection (2.2.6). In general the density function $p_\beta(\beta)$, (remembering that it is replaced by $p_\beta^*(\beta)$ defined by (2.55)), changes with each inspection. The probability of detection given by equation (2.106) accounts for only those structures that have cracks with lengths greater than a_d . Additional terms representing the effect of the removal function would have to be included.

The complete expression for $P_{\text{det}}(t_{i_j})$ has not been incorporated in the NERF computer program because equation (2.106) is used only when the simplifying approximation described in Section 3.6.2 is invoked. For the case of inspections involving the removal function, the probability of detection must be evaluated by taking the difference between $P_S(t)$ before and after the inspection.

3. DETAILED SPECIFICATIONS FOR THE MODELS EVALUATED BY NERF

In the previous Chapter, the mathematical basis for the most general reliability functions evaluated by the NERF computer program was presented. The resulting expressions, although detailed, require further processing before a complete numerical specification and description of the operation of NERF can be made.

There are two areas in which the general expressions are incomplete.

- (i) The effects of certain characteristics and/or limitations of the input data require explicit enunciation.
- (ii) A useful function of NERF is the evaluation of models which are simplifications of those described in the previous Chapter, either as the result of neglecting the variation in one or more of the basic random variables or by making approximations which significantly reduce the complexity of the analysis.

It is not necessarily true that expressions neglecting variations of a given random variable can be deduced in an obvious way from the most general expressions. It is therefore necessary to carefully examine each simplification and generate an appropriate set of reliability functions.

It is the function of this Chapter to complete the general expressions and provide specifications for all the facilities provided by the computer program.

3.1. Model Classification System

The NERF computer program has been written so that the variations of any combination of the basic random variables can be neglected from the analysis. If each possible combination is regarded as a separate model, 8 sets of reliability functions require specification. Moreover, the fact that zero initial crack length represents a special case of neglecting the variation of \underline{X}_2 , increases the model count to 12. Fortunately it is possible to classify the models in such a way that only 4 sets of reliability functions need specification.

The classification is made according to the status of the random variables \underline{X}_1 and \underline{X}_3 , representing comparative fatigue life and relative residual strength respectively. There are 4 possibilities, and for each model thus identified, the initial crack length random variable, \underline{X}_2 , can have status equal to one of the following:

- (i) $\bar{t}_1 < \underline{X}_2 < \bar{t}_f$ with density function $p_{\underline{X}_2}(x_2)$;
- (ii) $\underline{X}_2 = \text{constant}, \neq 0$;
- (iii) $\underline{X}_2 = 0$, corresponding to $n_0 \leq \bar{t}_1$.

Thus each class includes the two groupings identified in the previous Chapter, namely, initial cracking, (i) and (ii), and no initial cracking, (iii).

The general expressions derived in the previous Chapter all belong to the same class, (variations in both \underline{X}_1 and \underline{X}_3 included). Each reliability function for the full initial cracking model (variations in \underline{X}_2 included) can be written in the form

$$f(t) = \int_{n_1}^{n_2} p_{n_0}(n_0) f(t, n_0) n_0 \quad (3.1)$$

For constant n_0 , the same reliability function is represented by $f(t, n_0)$. If the definition of $f(t, n_0)$ includes extra terms representing the behaviour of uncracked structures when $n_0 = 0$, it can, together with expressions for n_1 and n_2 be used as a specification for that reliability function for the three states for \underline{X}_2 .

For example, the function

$$\begin{aligned} P_{\underline{S}}(t, n_0) = & \delta(n_0, 0) \int_{R_p}^{\infty} p_{\alpha}(\alpha) \exp\{-r_1(\alpha)t\} d\alpha P_{\beta}(\bar{t}_1) \\ & + \int_{t_1}^{t_2^*} P_{\beta}(\beta) \int_{\alpha}^{\infty} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \quad (3.2) \end{aligned}$$

where

$$H(\alpha, \beta, n_0, t) = \exp \left\{ -t/(\beta - n_0) \left[\delta(n_0, 0) r_1(\alpha) \bar{t}_1 + \int_{\max\{\bar{t}_1, n_0\}}^{\beta} r_2(\alpha, \beta', t) d\beta' \right] \right\} \quad \dots (3.3)$$

together with $n_1 = \bar{t}_1$ and $n_2 = \bar{t}_d$ provides a complete specification for $P_{\underline{S}}(t)$, replacing (2.59) and (2.60).

The reliability functions for a given class are defined in terms of functions for given n_0 as set out below.

(i) Probability of survival

$P_s(t, n_0)$ is found such that for the full initial crack model,

$$P_s(t) = \int_{n_1}^{n_2} P_s(t, n_0) p_{n_0}(n_0) dn_0. \quad (3.4)$$

(ii) Probability of failure

If $P_F(t, n_0)$ is the probability of failure for constant n_0 , that for the full initial crack model is,

$$P_F(t) = \int_{n_1}^{n_2} P_F(t, n_0) p_{n_0}(n_0) dn_0 + \bar{P}_{n_0}(n_2). \quad (3.5)$$

Note that the limits n_1 and n_2 may, in general, differ between functions.

(iii) Probability of detection

The probability of detection at constant n_0 is $P_{det}(t, n_0)$ so that,

$$P_{det}(t) = \int_{n_1}^{n_2} P_{det}(t, n_0) p_{n_0}(n_0) dn_0. \quad (3.6)$$

(iv) Risk rate

The risk rate is broken into 4 components;
 the risk of static failure by fatigue $r_s(t, n_0)$,
 the risk of fatigue life exhaustion (minimum strength) $r_{f1}(t, n_0)$,
 the risk of fatigue life exhaustion (life limit) $r_{f2}(t, n_0)$
 and the virgin risk $r_v(t)$.

The first three components bear the same relationships to those for the full initial crack model as expressed by equation (3.1). The fourth is valid only for $n_0 = 0$.

The density function for the time to failure can also be split into four components,

$$p_{fs}(t, n_0) = \underline{P}_s(t) \cdot r_s(t, n_0), \quad (3.7)$$

$$p_{ff1}(t, n_0) = \underline{P}_s(t) r_{f1}(t, n_0), \quad (3.8)$$

$$p_{ff2}(t, n_0) = \underline{P}_s(t) r_{f2}(t, n_0) \quad (3.9)$$

$$\text{and } p_{fv}(t) = \underline{P}_s(t) r_v(t). \quad (3.10)$$

In equations (3.7 - 3.10) $\underline{P}_s(t)$ is the probability of survival relevant to the current n_0 status, i.e. if n_0 is constant $\underline{P}_s(t) = P_s(t, n_0)$.

In practice, integral expressions for $p_{fs}(t, n_0)$, $p_{ff1}(t, n_0)$, $p_{ff2}(t, n_0)$ and $p_v(t)$ are obtained. The component risk rates can be obtained via equations (3.7 - 3.10). The total

risk rate is then given by,

$$r(t) = \int_{n_1}^{n_2} p_f(t, n_0) dn_0 / P_S(t) \quad (3.11)$$

where

$$p_f(t, n_0) = \delta(n_0, 0)p_{fv}(t) + p_{fs}(t, n_0) + p_{ff1}(t, n_0) + p_{ff2}(t, n_0). \quad (3.12)$$

(v) Density for strength

$$1f F_R(R, t, n_0) = P_S(t)p(R|\underline{F} > t, n_0) \quad (3.13)$$

then

$$p_R(R|\underline{F} > t) = \int_{n_1}^{n_2} f_R(t, n_0)p_{n_0}(n_0)dn_0 / P_S(t) \quad (3.14)$$

(iv) Failure density for strength

Although an equation similar to (3.14) can be written for $p_R(R|t)$, an alternative, in terms of functions already defined above can be derived. This alternative expression leads to computational efficiency by reducing the number of integrations that have to be performed to generate a complete set of reliability functions.

Defining $f_{R,f2}(R,t,n_0)$ to be a function such that

$$p_{ff2}(t,n_0) = \int_{R_{\min}}^{\infty} f_{R,f2}(R,t,n_0) dR, \quad (3.15)$$

$$\begin{aligned} p_f(t)p_{\underline{R}}(R|t) &= f_{\underline{R}}(R,t,n_0)r_2(R) + \delta(R,R_{\min})p_{ff1}(t,n_0) \\ &\quad + f_{R,f2}(R,t,n_0) \\ &= \xi_R(t,n_0) \text{ say.} \end{aligned} \quad (3.16)$$

Then for the full initial crack model,

$$p_{\underline{R}}(R|t) = \int_{n_1}^{n_2} \xi_R(t,n_0)p_{n_0}(n_0)dn_0/p_{\underline{f}}(t) \quad (3.17)$$

For a given class, the 9 functions, $P_{\underline{S}}(t,n_0)$, $P_{\underline{F}}(t,n_0)$, $P_{\det}(t,n_0)$, $p_{fs}(t,n_0)$, $p_{ff1}(t,n_0)$, $p_{ff2}(t,n_0)$, $p_{fv}(t,n_0)$, $f_{\underline{R}}(R,t,n_0)$ and $f_{R,f2}(R,t,n_0)$ are defined by integral expressions. All other reliability functions can be expressed in terms of these nine. The limits n_1 and n_2 are required, in the case of the full initial crack model, for the functions $P_{\underline{S}}(t)$, $P_{\underline{F}}(t)$, $P_{\det}(t)$, $r_s(t)$, $r_f(t)$ ($= r_{f1}(t) + r_{f2}(t)$), $p_{\underline{R}}(R/\underline{F}|t)$ and $p_{\underline{R}}(R|t)$.

3.2. Input Data Specifications

Before the classification method described in the previous section can be applied to the general reliability functions, it is appropriate to consider the properties of the data upon which the reliability analysis will be based. In particular, limits imposed by the data may have repercussions in terms of the equations defining the limits of integration in the various reliability expressions.

This consideration is presented here in the form of specifications for the input data for the NERF computer program. In addition to the specifications, example data will be described. These data define two populations of structures and applied load sequences which will be used to demonstrate various aspects of the methodology and computer coding as the need arises.

The first set of data pertain to an idealized set of structures: the input functions have relatively simple forms. This set, referred to as 'example A' will be used predominately for models without initial cracks.

The second set of data is more typical of an aircraft fleet and will be used as input data for initial crack models. It will be referred to as 'example B'.

3.2.1. Crack growth function

The relationship between crack length and time is assumed to be specified by a function which represents the average behaviour of the population of structures. This function,

$$a = a(\tilde{t}) \quad (3.18)$$

is defined for values of the argument, median time or age, in the interval $[\tilde{t}_1, \tilde{t}_f]$ where \tilde{t}_1 is the median initiation time and \tilde{t}_f the median fatigue life for the population.

It is an important feature of the NERF computer program that this function, together with other input functions, is defined by a set of ordered pairs of argument and function value without recourse to functional forms. The ordered pairs are assumed to be such that a cubic spline fitted through them defines the continuous curve representing the function to an acceptable degree of precision.

The crack growth functions for the two examples are shown in Figures 3.1 and 3.2. The data for example A was obtained by assuming that (3.18) was the solution of,

$$\frac{da}{d\tilde{t}} = Ca^3 \quad (3.19)$$

such that when $\tilde{t} = \tilde{t}_1 = 400$ hrs $a = 0.1$ mm; when $\tilde{t} = \tilde{t}_f = 4200$ hrs $a = 0.41$ mm. The resulting function is,

$$a = 6.164 / (4176 - 0.94\tilde{t})^{\frac{1}{2}} \quad (3.20)$$

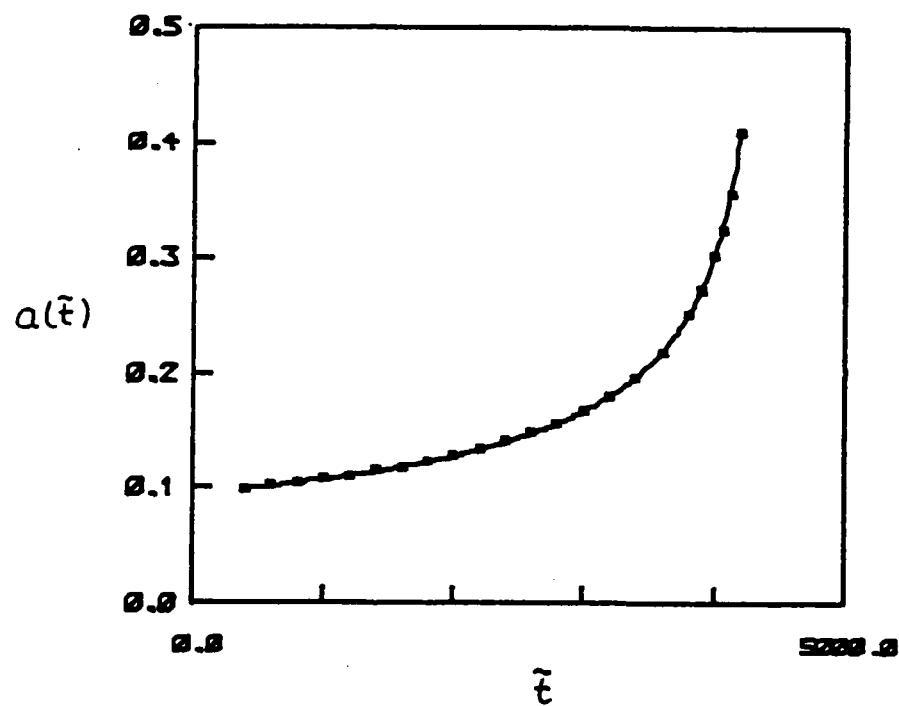


Figure 3.1. Crack growth function, $a = a(\tilde{t})$ for example A.

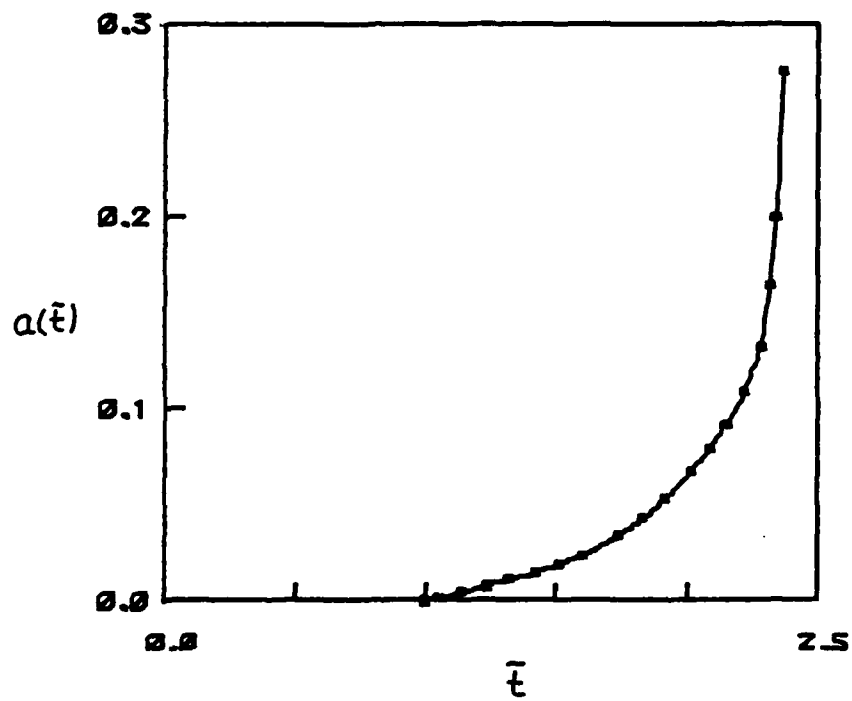


Figure 3.2. Crack growth function, $a = a(t)$ for example B.

The data points indicated in Figure 3.1. are defined by equation (3.20): the curve is the cubic spline fitted by NERF through the points. Note that, as illustrated by this example, the initial crack length $a(\tilde{t}_1)$ can be non-zero.

The data for example B represent a 'best fit' curve for measured crack growth. It is not necessary that the precise form of the function is known.

A requirement of the reliability functions is that the inverse crack growth function is available, i.e.,

$$\tilde{t} = a^{-1}(a) \quad (3.21)$$

for $a_1 \leq a \leq a_f$, where $a_1 = a(\tilde{t}_1)$ and $a_f = a(\tilde{t}_f)$. This function is obtained by solving (3.18) for \tilde{t} , given a . Inverse interpolation is not used because, in general, the inverse interpolating function is not identical with the forward interpolating function. The procedure used by NERF ensures that the difference between a given value of \tilde{t} and $a^{-1}(a(\tilde{t}))$ can be made to be insignificant.

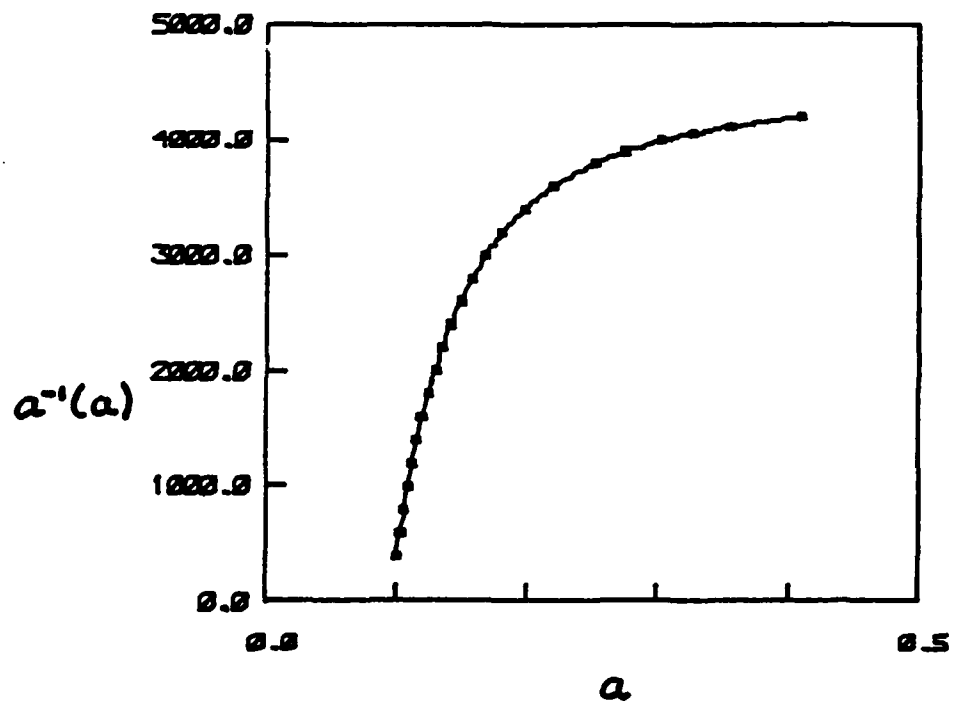


Figure 3.3. Inverse crack growth function for example A.

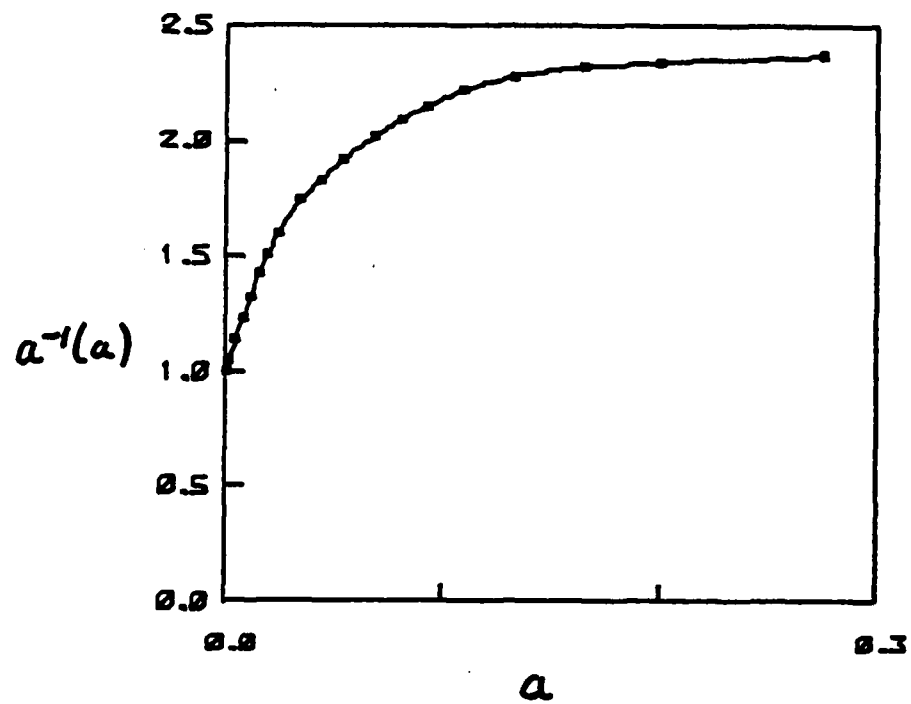


Figure 3.4. Inverse crack growth function for example B.

3.2.2. Strength decay function

The relationship between crack length and strength is assumed to be defined for the median structure, viz.,

$$\tilde{R} = \tilde{R}(a) \quad (3.22)$$

over the interval $[a_1, a_2]$ which does not necessarily coincide with $[a_1, a_f]$. For $a < a_1$, $\tilde{R} = \tilde{R}(a_1)$ and for $a > a_2$, $\tilde{R} = \tilde{R}(a_2)$.

As was the case for the crack growth function, the strength decay function is defined by a set of ordered pairs which are used by NERF to establish a cubic spline representation for $\tilde{R}(a)$.

The strength decay function for example A is shown in Figure 3.5. This function is of the form,

$$\tilde{R} = c_1 \cdot a^{-\frac{1}{2}} \quad (3.23)$$

with c_1 chosen so that $\tilde{R}(a_1) = 100$, i.e.,

$$\tilde{R} = 31.623 a^{-\frac{1}{2}} \quad \text{for } a \in [0.1, 1]. \quad (3.24)$$

The corresponding function, derived from experimental data, for example B is shown in Figure 3.6.

The strength decay function is used to generate the relative strength function, $\psi(\tilde{t})$, (equation 2.25). This function is defined by a set of ordered pairs of \tilde{t} and $\psi(\tilde{t})$ where the values of \tilde{t} are the same as those used to define the crack growth function. $\psi(\tilde{t})$ is thus defined over the

75

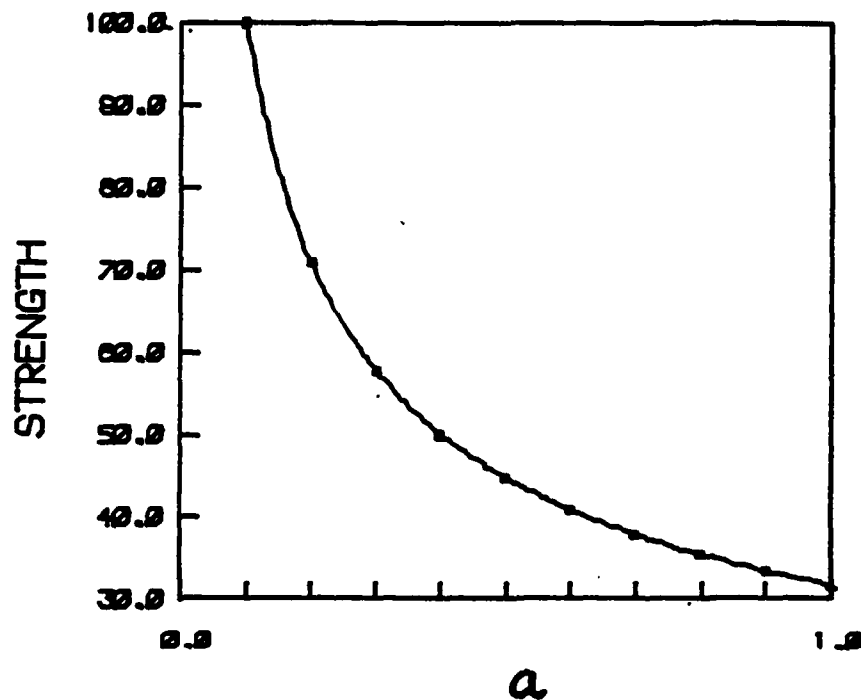


Figure 3.5. Strength decay function, $R = R(a)$ for example A.

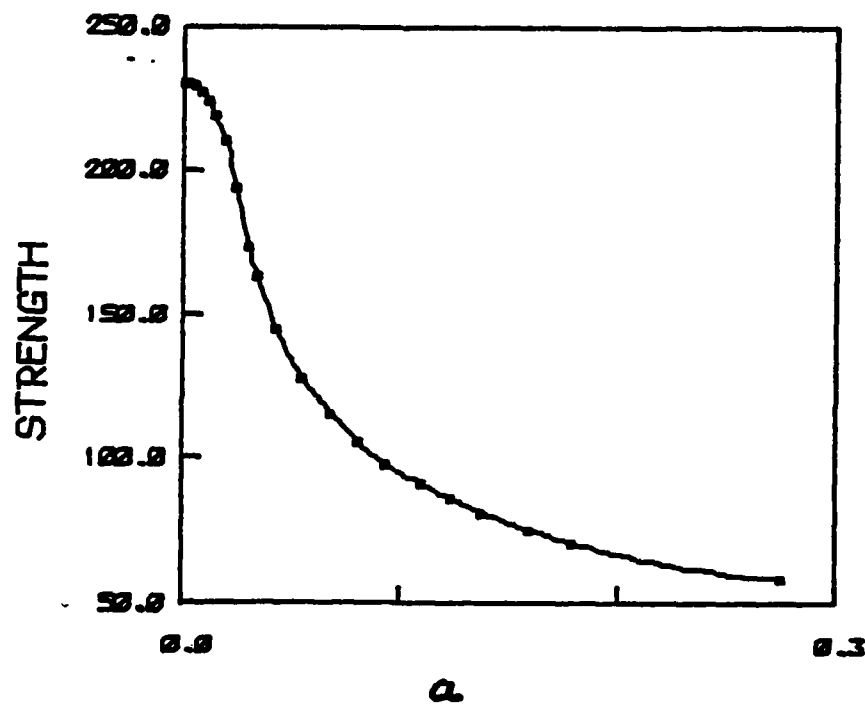


Figure 3.6. Strength decay function, $R = R(a)$, for example B.

interval $[\tilde{t}_1, \tilde{t}_f]$. For $\tilde{t} < \tilde{t}_1$, $\psi = 1$ and for $\tilde{t} > \tilde{t}_f$, $\psi = 0$.

The derived relative strength functions for the two examples are shown in Figures 3.7 and 3.8. Note that the function for example B has nearly zero first derivative for $\tilde{t} \sim \tilde{t}_1$. In order to prevent numerical problems when defining the inverse function, $\psi^{-1}(\psi)$, linear interpolation is used for $\tilde{t}_1 \leq \tilde{t} \leq \tilde{t}_j$ where \tilde{t}_j is \tilde{t} value of the last definition point (or node) for which $\psi \geq 0.9$.

An inverse function is defined, either by linear interpolation for $\tilde{t}_1 \leq \tilde{t} \leq \tilde{t}_j$ or by solving the equation $\psi = \gamma(\tilde{t})$ for \tilde{t} using the cubic spline for $\tilde{t} > \tilde{t}_j$. For $\psi > 1$, $\tilde{t} = \tilde{t}_1$ and for $\psi < \gamma(\tilde{t}_f)$, $\tilde{t} = \tilde{t}_f$.

The derivative, $\frac{d\psi}{d\tilde{t}}$ is generated by differentiating the interpolation function (either linear or cubic spline) directly. For $\tilde{t} < \tilde{t}_1$ $\psi' = 0$ and for $\tilde{t}_f < \tilde{t}$ $\psi' = 0$.

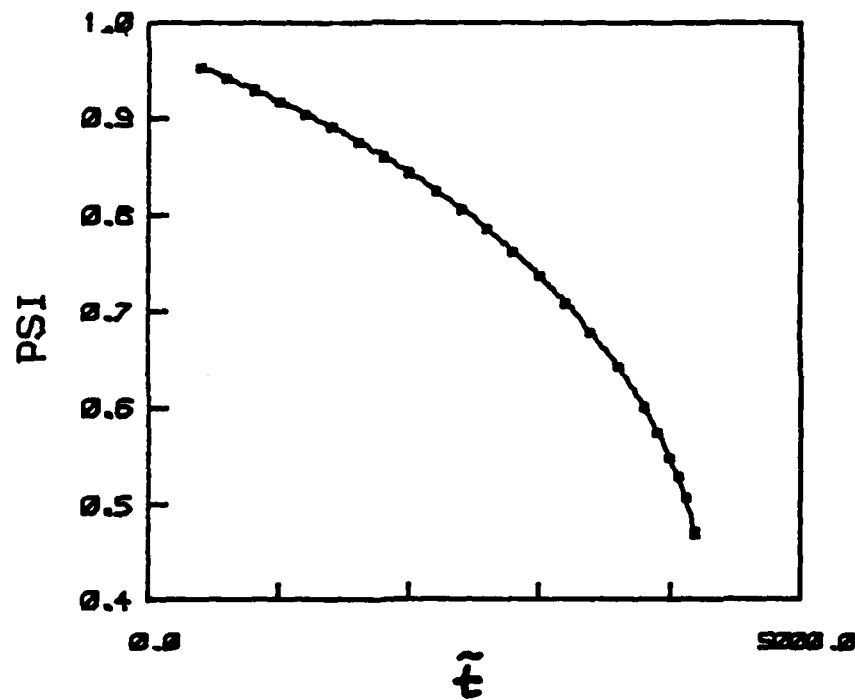


Figure 3.7. Relative strength decay function,
 $\psi = \psi(\tilde{t})$ for example A.

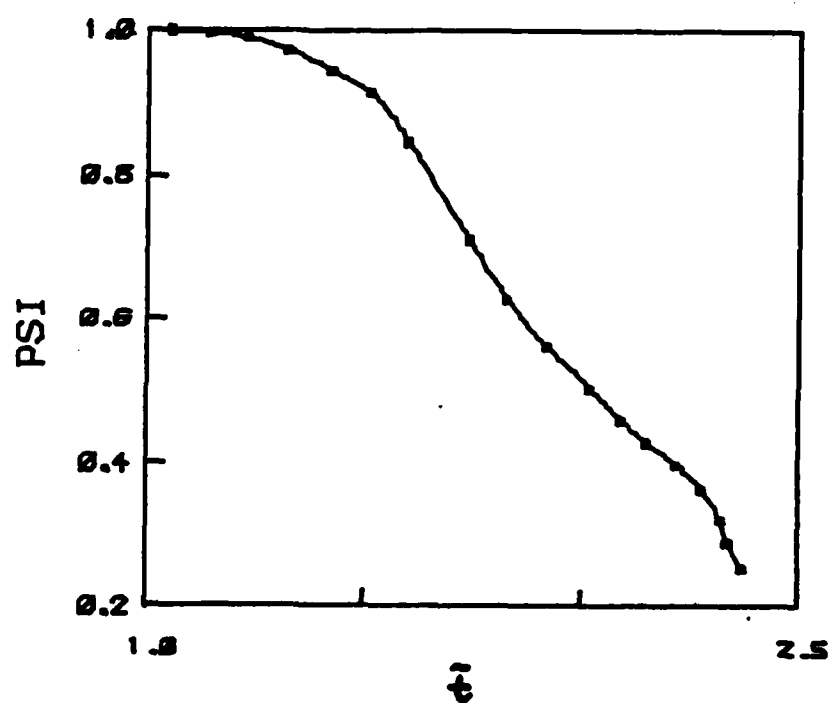


Figure 3.8. Relative strength decay function,
 $\psi = \psi(\tilde{t})$, for example B.

3.2.3. Probability of load exceedence

The risk rate for a structure of given strength, R , is related to the probability of load exceedence, defined by equation (2.20), viz,

$$r_k(R) = \bar{P}_L(R) l_r. \quad (3.25)$$

The specification for $\bar{P}_L(R)$ can also be regarded as a specification for $r_k(R)$: if $l_r = 1$, the two functions are identical. The function $P_L(R)$ is assumed to be defined by a set of ordered pairs of R and $\bar{P}_L(R)$ for $R_{\min} \leq R \leq R_{\max}$. R_{\min} is the first node for $\bar{P}_L(R)$ and defines the minimum value of strength. R_{\max} is the maximum value of load in the applied sequence and corresponds to the last node in the definition of $\bar{P}_L(R)$.

The function $\bar{P}_L(R)$ and the load application rate l_r are used by NERF to define $r_k(R)$, (sometimes referred to as the load exceedence rate) via equation (3.25). Interpolation is obtained by taking the natural logarithm of $r_k(R)$ for each of the nodes and fitting a cubic spline through the resulting ordered pairs.

The probability of load exceedence for example A is,

$$\bar{P}_L(R) = \exp \{-2.303(R - 30)\} \quad (3.26)$$

corresponding to a density function for load application,

$$p_L(R) = 2.303 \exp \{-2.303(R - 30)\} \quad (3.27)$$

for $30 \leq R \leq 110$. With $l_r = 10$,

$$r_k(R) = 10 \exp \{-2.303(R - 30)\}. \quad (3.28)$$

This function is shown in Figure 3.9. The nodes are at $R = 30$ (10) 110 and the interpolating polynomial is, of course, linear for $\log(r_k(R))$. Note that although the interpolation within NERF uses natural logarithms, the functions are plotted after taking logarithms to base 10.

A typical function based on experimental data is used for example B and is shown in Figure 3.10. Note that the distribution deviates from exponential for small values of R only. For this function $l_r = 500\ 000$.

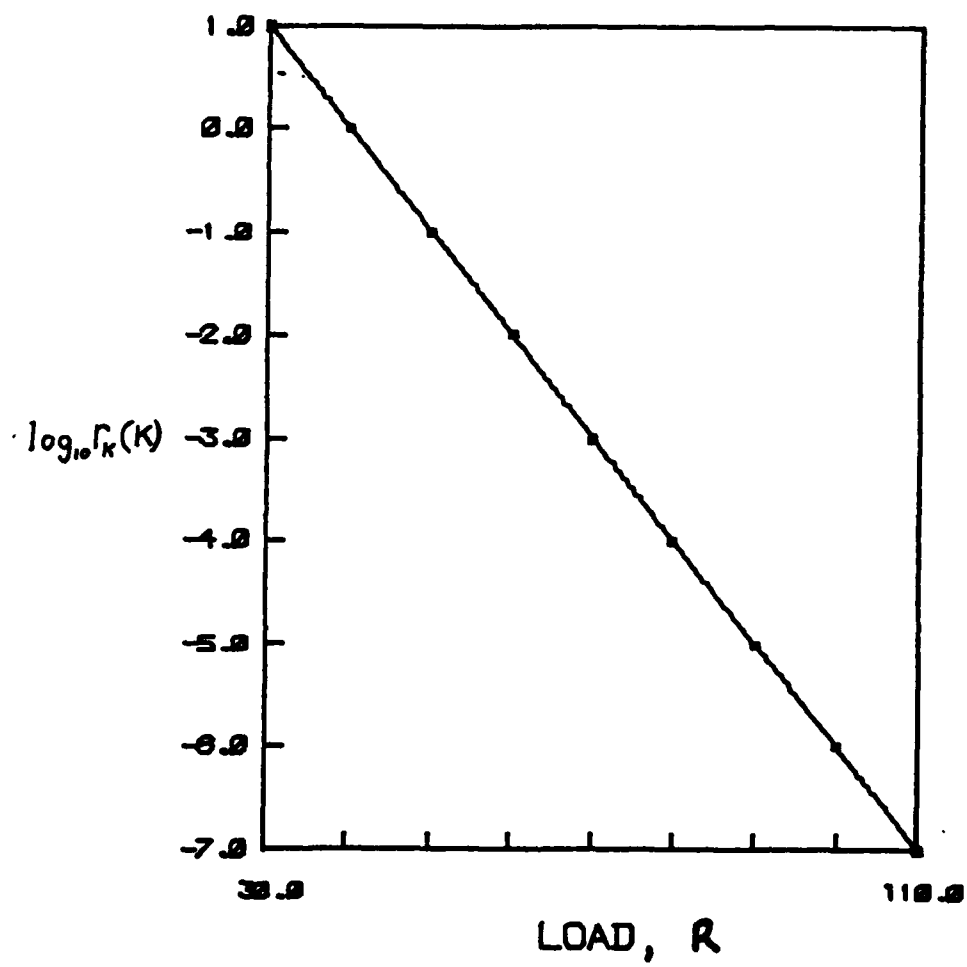


Figure 3.9. Risk rate, $r_K r_K(R) = \bar{F}_K(R) \cdot 1_r$ for example A.

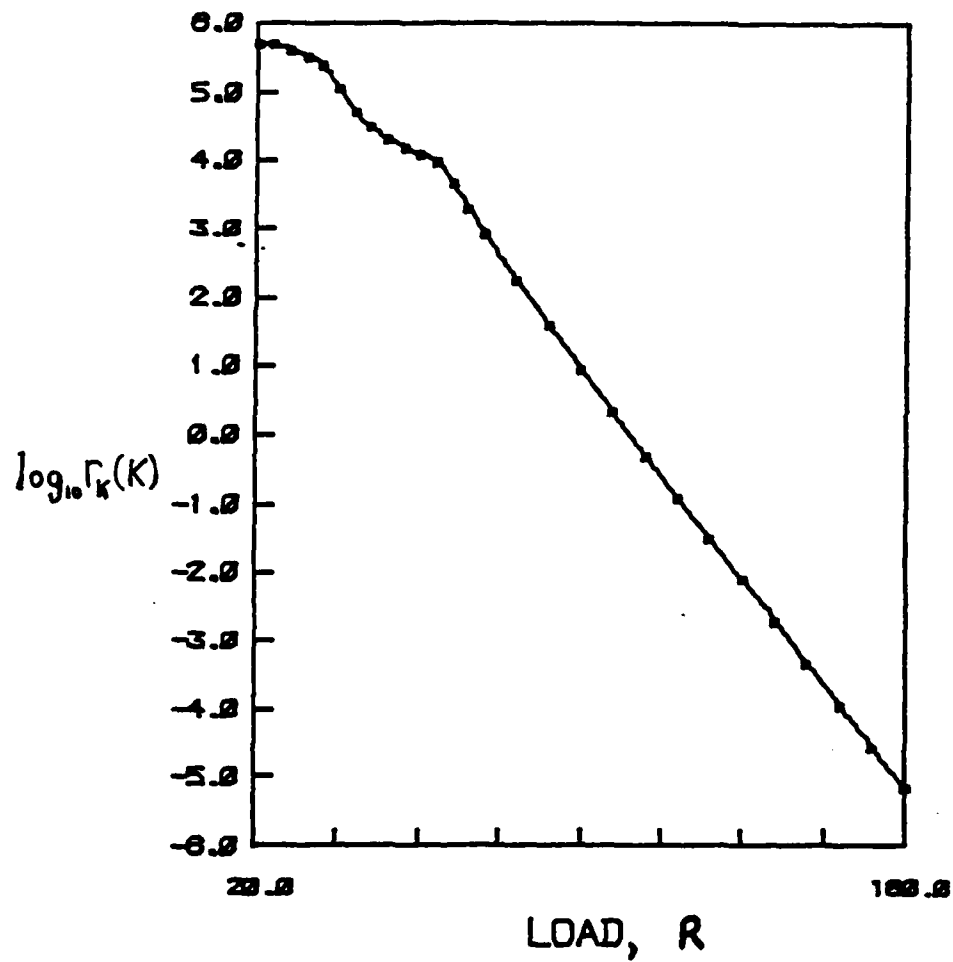


Figure 3.10. Risk rate, $r_k = r_k(R) = P_L(R) \cdot l_r$ for example B.

3.2.4. Inspection removal and crack detection functions

The crack length inspection removal function $S(\alpha, \beta, t, t_{i_j})$, (Section 2.2.6), defined by equation (2.52) can be written in the form

$$S(\alpha, \beta, n_0, t, t_{i_j}) = H_g((\tilde{t}_d - n_0)t/t_{i_j} + n_0 - \beta) \cdot S_j(\tilde{t}) \quad (3.29)$$

where

$$\tilde{t} = n_0 + (\beta - n_0)t_{i_j}/t. \quad (3.29a)$$

The effect of the unit step function is taken into account by the integration limits in the expressions for the reliability functions. The function $S_j(\tilde{t})$ which remains invariant with respect to the inspection sequence is specified by defining a function $C_d(a)$ called the crack detection function. This function is specified by a sequence of ordered pairs with a in the interval $[0, a_d]$ where a_d is the crack length beyond which the inspection is assumed to be perfect. $C_d(a)$ has the general form shown in Figure 3.11.

$C_d(a)$ is analogous, in fact, to a cumulative probability of detection for crack length. The term crack detection function has been used here to remove confusion with the probability of detection defined in Section 2.2.11.

NERF makes the conversion from $C_d(a)$ to $S_j(t)$ via the relationship,

$$S_j(\tilde{t}) = 1 - C_d(a(\tilde{t})) \quad (3.30)$$

so that $S_j(\tilde{t})$ has the general form shown in Figure 3.12.

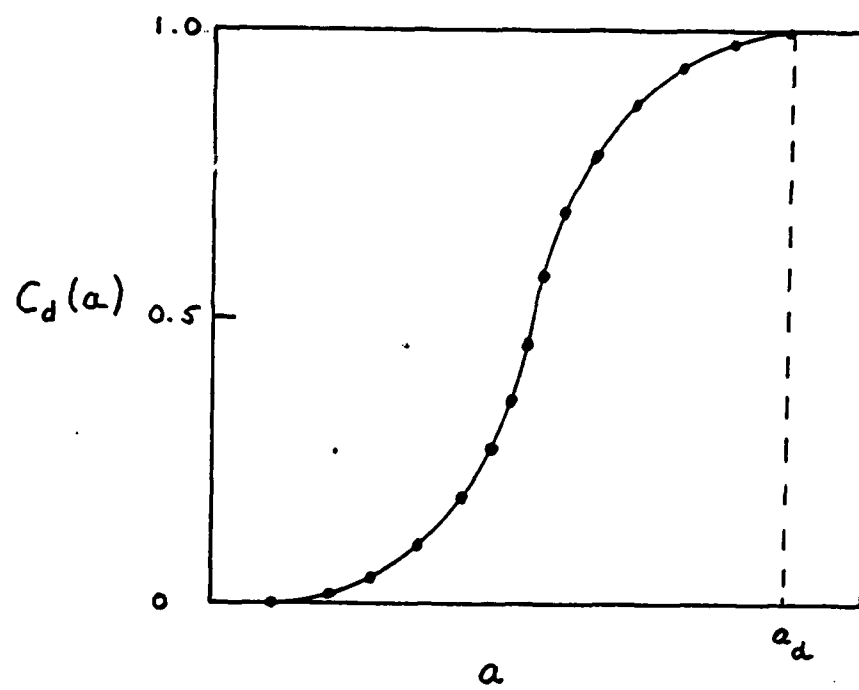


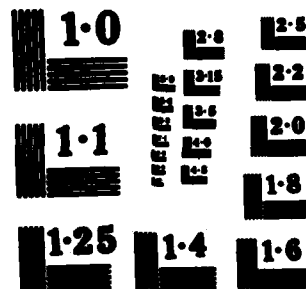
Figure 3.11. General form of the crack detection function.

[illegible]

(OF RELIABILITY FUN. (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARI/STRUC-397 F/G 9/

2/7

NI



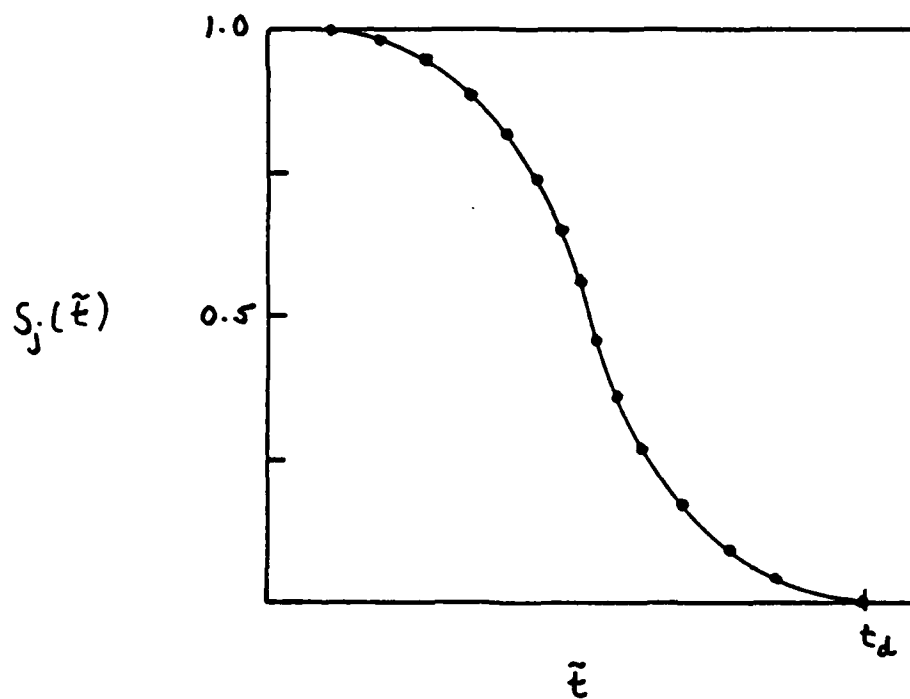


Figure 3.12. General form of the crack length inspection removal function. This function is derived from the crack detection function and the nodal values of \tilde{t} correspond to those of a used to defined $C_d(a)$.

The function $S_j(\tilde{t})$ is thus defined by a sequence of ordered pairs of \tilde{t} and S_j , where the values of \tilde{t} correspond to the values of a used to define $C_d(a)$.

The definition of $C_d(a)$ is used by NERF to set values for a_d and $\tilde{t}_{d,}(a^{-1}(a_d))$. Provision is made for a_d to be specified independently, in which case the inspection process is assumed to be perfect and the specification of $C_d(a)$ is ignored.

Note that $C_d(a)$ need be defined over only a limited range of a . For values of a outside and less than the lower limit of this range $C_d(a)$ is assumed to be zero.

3.2.5. Density functions for the basic random variables

The density functions for the basic random variables (comparative fatigue life, initial crack length and relative residual strength) are assumed to be continuous and dependent on at most three parameters in the following way.

NERF provides a library of standard probability density functions which are defined by a single dispersion parameter for a standard random variable, \underline{z} say. Currently this library contains three density functions:

(i) log normal,

$$p_{\underline{z}}(z) = \frac{1}{z\sqrt{2k\pi}} \exp\left\{-\frac{1}{2}[\log(z)/k]^2\right\} \quad ; \quad (3.31)$$

(ii) extreme value,

$$p_{\underline{z}}(z) = kz^{k-1} \exp\{-z^k\} \quad \text{and} \quad (3.31a)$$

(iii) gamma,

$$p_{\underline{z}}(z) = z^{k-1} \exp\{-z\} / \Gamma(k) \quad . \quad (3.31b)$$

In each function, k is the dispersion parameter.

Each basic random variable is related to the standard random variable by an equation of the form,

$$\underline{z} = (\underline{x} - e)/v \quad (3.32)$$

where v and e are called the location and minimum value parameters respectively.

For each basic random variable, the density function is completely specified by selecting the appropriate standard function and defining the dispersion, location and minimum value parameters. The specifications for the two example problems are presented in table 3.1.

Of importance, in so far as the reliability functions are concerned, is that each density function will be significant over a restricted range of its argument. Normally the range of a random variable will be found by NERF and will be defined by the values beyond which the density is less than some small value, typically 10^{-8} . Alternatively, the range may be restricted as part of the definition of the input data. This is particularly true for initial crack length when the range is restricted by some inspection procedure prior to the commencement of life. For such cases, the density function is normalised, thus for X_2 say,

$$p_{X_2}(x_2) = \frac{p'_{X_2}(x_2)}{\int_{x_{2,\min}}^{x_{2,\max}} p'_{X_2}(x_2) dx_2} \quad (3.33)$$

where $p'_{X_2}(x_2)$ is the density function defined by the input parameters. and $x_{2,\min}$ and $x_{2,\max}$ are the defined extremities of the range for X_2 .

(Note that the current version of the computer program applies this normalising procedure to the density function for initial crack length only.)

	Random Variable	Type	Dispersion (k)	Location (v)	Min value (e)
A	Comparative Fatigue Life	log normal	0.2	0.0	0.0
	Relative Residual Strength	extreme value	10.0	1.0	0.2
B	Comparative Fatigue Life	log normal	0.2	0.0	0.0
	Relative Residual Strength	extreme value	6.0	1.0	0.6
	Initial Crack Length	Gamma	3.0	120.0	1.0

Table 3.1. Specifications for the density functions used in the examples.

3.2.6. Model parameters

The reliability functions depend on several parameters, in addition to those which define the probability density functions, some of which are defined as separate parameters and some as integral parts of the median functions. The various parameters and their methods of definition are described below.

(1) \tilde{R}_0 : Median virgin strength

The median virgin strength is an independent parameter for the analysis and has the effect of specifying the transformation from \underline{X}_3 , (relative residual strength), to α , (virgin strength), i.e.

$$\alpha = \tilde{R}_0 \underline{X}_3. \quad (3.34)$$

Equation (3.30) allows the density function for virgin strength to be defined,

$$p_\alpha(\alpha) = p_{\underline{X}_3}(\alpha/\tilde{R}_0)/\tilde{R}_0 \quad (3.35)$$

If the variation in \underline{X}_3 is neglected, \tilde{R}_0 defines the virgin strength.

(11) l_r : Average load application rate

The average load application rate is an independent parameter which completes the specification of the relationship between the probability of load exceedance and the risk rate, equation (3.25). In effect, l_r determines the time scale for the reliability model.

(iii) R_{\min} : Minimum value of strength

R_{\min} is defined by the first node of the sequence of ordered pairs defining the probability of load exceedance. It is interpreted by NERF as the minimum value of strength below which structures can not exist.

(iv) R_{\max} : Maximum value of strength subject to risk

R_{\max} is defined by the last node of the probability of load exceedance. For $R > R_{\max}$, $r_k = 0$. This parameter has the effect of limiting the integration domains for the reliability functions.

(v) R_p : Proof load limit

R_p is an independent parameter which defines the criterion for proof load inspections where all structures with $R \leq R_p$ are removed. If $R_p = 0$, no proof load inspections occur.

(vi) \tilde{t}_1 : Median crack initiation time

\tilde{t}_1 is defined by the first node in the crack growth function and usually denotes the defined median crack initiation time. However, NERF assumes the added interpretation that \tilde{t}_1 denotes the age at which structures start to experience a risk as a result of cracking. For example, if $X_3 = \tilde{R}_0$ and $\tilde{R}_0 > R_{\max}$, structures will commence experiencing a risk rate for $\tilde{t} > \psi^{-1}(R_{\max}/\tilde{R}_0)$. This defines an effective value for \tilde{t}_1 which overrides a smaller defined value.

(vii) \tilde{t}_f : Median fatigue life

\tilde{t}_f is defined by the last node in the crack growth function and determines the median limit for age. All structures which reach an age equal to \tilde{t}_f are deemed to have failed. The question of fatigue life limiting is considered in detail in the following Section.

(viii) $a_{0,max}$: Maximum initial crack length

For a model in which a_0 (or n_0) is allowed variation, $a_{0,max}$ is an independent parameter which specifies the upper limit for initial crack length.

If a_0 is not allowed variation, $a_{0,max}$ determines the constant value of a_0 and n_0 via $n_0 = a^{-1}(a_{0,max})$. If $a_{0,max} = 0$, NERF selects the non-initial crack version of the current class. (Note that $n_0 \leq \tilde{t}_1$ corresponds to $a^{-1}(0)$ so that a non-initial crack model actually corresponds to $n_0 \leq \tilde{t}_1$ rather than just $n_0 = 0$.)

(ix) a_d : Crack length inspection criterion

a_d is an independent parameter which defines the criterion for a crack length inspection. The age limit used in the models is \tilde{t}_d where

$$\tilde{t}_d = a^{-1}(a_d) . \quad (3.36)$$

For a model with no crack length inspections, $\tilde{t}_d \geq \tilde{t}_f$.

If a_d is specified as zero, then a crack detection function must be defined.

A summary of the model parameters, their definitions and values for the two examples is presented in Table 3.2.

Symbol	Name	Definition	Example A	Example B
R_0	Median virgin strength	Independent	100	230
l_r	Average load application rate	Independent	10	500 000
R_{min}	Minimum value of strength	First node of $P_L(R)$	30	20
R_{max}	Maximum value of strength subject to risk	Last node of $P_L(R)$	110	180
R_p	Proof load limit	Independent		
t_1	Median crack initiation time	First node of $a(t)$	400	1
t_f	Median fatigue life	Last node of $a(t)$	4200	2.37
$a_{0,max}$	Maximum initial crack length	Independent		0.05
a_d	Crack length inspection criterion	Independent or last node of $C_d(a)$.		

Table 3.2. Summary of model parameters and their values in the two sets of example data.

3.2.7. Fatigue life limiting

An assumption made by all the reliability models evaluated by NERF is that each structure has a fatigue life limit given by the time of transition from D_2 (cracked time zone) to D_3 (failed time zone). The model equations developed in Section 2.2.3 can be used to derive an expression for this limit, t_f , by recognising that it corresponds to the instant when the structure has either reached an age of t_f , or has had its strength reduced to R_{min} , so that, (using (2.23)),

$$t_f = x_1(\min\{\tilde{t}_f, \psi^{-1}(R_{min}/x_3\tilde{R}_0) - a^{-1}(x_2)\}) \quad (3.37)$$

Any structure reaching this fatigue life limit contributes to the risk of fatigue life exhaustion, r_f .

An observation that can be made from equation (3.37) is that structures with $x_3 > R_{min}/\tilde{R}_0\psi(\tilde{t}_f)$ have fatigue life limits that are independent of the strength random variable, \tilde{x}_3 . For these structures the fatigue life limit can be interpreted as the time of instantaneous strength decay (or 'runaway' crack growth). However, this interpretation assumes that only the stronger structures experience runaway cracking which may not be the case in general. In fact, placing such detailed interpretations on the modes of failure of structures reaching their fatigue life limits is open to argument and the model needs careful examination in cases where the risk of fatigue life exhaustion is dominant.

The significance of fatigue life limiting for a given set of data can be estimated, in the mean, by computing the loss factor at \bar{t}_f for a population without variations, i.e.,

$$\begin{aligned}
 H(1, \bar{t}_f, 0, \bar{t}_f) &= \exp \left\{ - \left[r_1(1) \bar{t}_1 + \int_{\bar{t}_1}^{\bar{t}_f} r_2(1, \beta') d\beta' \right] \right\} \\
 &= \exp \left\{ -l_r \left[\bar{P}_L(\bar{R}_0) \bar{t}_1 + \int_{\bar{t}_1}^{\bar{t}_f} \bar{P}_L(\bar{R}_0 \psi(\beta')) d\beta' \right] \right\} .
 \end{aligned}$$

.... (3.38)

This expression represents the fraction of a uniform population having the median structural properties that would survive until \bar{t}_f under the influence of the load sequence represented by $P_L(R)$. If this fraction is small (say $< 10^{-4}$) then the risk of fatigue life exhaustion will be an insignificant part of the total risk. However, if this term is ~ 1 then the model will be dominated by failures due to fatigue life limiting. Such a case will probably represent a load spectrum which is too weak or, as equation (3.38) indicates, a load application rate, l_r , which is too small.

3.3. Data Limits and Their Effects

The integration limits appearing in the expressions developed in Chapter 2 result from assumptions inherent in the reliability model. In practice, the integration domains defined by these limits may include regions where the integrand is insignificant. Although such regions present no essential mathematical problems, improved performance of the numerical integration algorithms is achieved if the integration limits are modified to explicitly exclude these regions. The numerical requirement for explicit exclusion of insignificant regions is particularly important in those cases when the boundaries appear as discontinuities in the integrand.

Modifications to the integration limits arise from the following limitations imposed by the data specified in Section 3.2.

(1) Truncation of load exceedence probability

$\bar{P}_L(R)$ is defined for $R_{\min} \leq R \leq R_{\max}$. The lower bound for R is already incorporated in the reliability expressions via the fatigue life limit assumptions. The upper limit must be incorporated in expressions where the integrand includes $\bar{P}_L(R)$ as a product term as in the calculation of $r_s(t)$, (e.g. equation 2.75).

(ii) Restriction of random variable ranges

The range of validity for each random variable is restricted, either by truncation or by the fact that the associated probability density function is significant over only a restricted range of its argument. Accordingly the basic random variables are assumed to be limited by

$$x_{i,\min} \leq x_i \leq x_{i,\max} \quad (i=1,2,3) \quad (3.39)$$

from which limits for the transformed random variables can be found, i.e.,

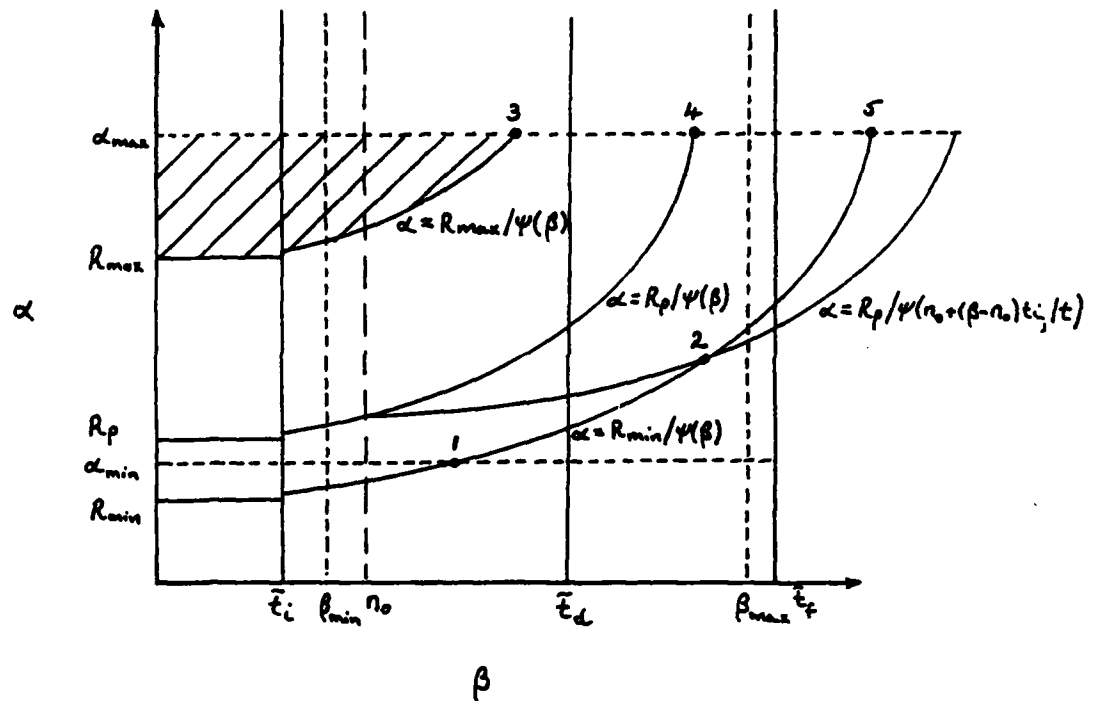
$$\alpha_{\min} \leq \alpha \leq \alpha_{\max} , \quad (3.40)$$

$$\beta_{\min} \leq \beta \leq \beta_{\max} , \quad (3.41)$$

$$n_{0,\min} \leq n_0 \leq n_{0,\max} . \quad (3.42)$$

The limits for n_0 can be readily included in the expressions for n_1 and n_2 (see equation 3.1) for each reliability function. The effects of the limits imposed on α , β and R can be understood by constructing their positions in the (α, β) plane for constant n_0 . Typical juxtapositions of the limit boundaries are shown in figure 3.13 which now replaces figure 2.5 as the basis for the calculation of the reliability functions following a combined proof load and crack length inspection.

To cope with the seemingly complex interaction of the integration boundaries, the following conventions regarding notation and default limiting conditions are adopted.



1. $\alpha = \alpha_{\min}; \beta = \psi^{-1}(R_{\min}/\alpha_{\min})$
 2. $\alpha = \alpha_{p, R_{\min}}; \beta = \beta_{p, R_{\min}}$
 3. $\alpha = \alpha_{\max}; \beta = \psi^{-1}(R_{\max}/\alpha_{\max})$
 4. $\alpha = \alpha_{\max}; \beta = \psi^{-1}(R_p/\alpha_{\max})$
 5. $\alpha = \alpha_{\max}; \beta = \psi^{-1}(R_{\min}/\alpha_{\max})$
- // Excluded from integration.

Figure 3.13. Typical juxtaposition of the various limit boundaries and the subspace boundaries.

3.3.1. Default limits

Despite the apparent freedom with which the various parameters in the model can be specified, there are certain limiting relationships implied by the model equations. Using primes to denote input data specifications the following sequence of equations exemplifies these relationships and defines new values for the various parameters. Throughout the remainder of this document reference to these parameters will imply the limiting relationships.

$$R_{\max} = \min\{\alpha_{\max}, R'_{\max}\}, \quad (3.43)$$

$$R_{\min} = \max\{\alpha_{\min} \psi(\tilde{t}'_f), R'_{\min}\}, \quad (3.44)$$

$$\tilde{t}_1 = \max\{\tilde{t}'_1, \psi^{-1}(R_{\max}/\alpha_{\min})\}, \quad (3.45)$$

$$\tilde{t}_f = \min\{\tilde{t}'_f, \psi^{-1}(R_{\min}/\alpha_{\max})\} \quad (3.46)$$

$$\tilde{t}_d = \min\{\tilde{t}'_d, \psi^{-1}(R_p/\alpha_{\max})\} \quad (3.47)$$

$$n_{0,\min} = \max\{\tilde{t}_1, n'_{0,\min}\} \quad (3.48)$$

$$\begin{aligned} n_{0,\max} &= \min\{\tilde{t}_f, n'_{0,\max}\} && \text{before 1st inspection,} \\ &= \min\{\tilde{t}_d, n'_{0,\max}\} && \text{after 1st inspection.} \end{aligned}$$

$$\dots \quad (3.49)$$

3.3.2. Notation for limit functions

The following functions are used to denote integration limits determined by the various boundaries illustrated in Figure 3.13. These functions encompass the various parameters used in Chapter 2 and have been generalized to allow for the effects of reductions in the number of random variables in the model as detailed later in this Chapter.

The functions are used primarily to delineate the D_2 subspace for cracked structures in α, β space given n_0 .

- (i) $\tilde{t}_f^*(R)$: Upper β bound for a domain bounded below by $R = \alpha \psi(\beta)$.

$$\tilde{t}_f^*(R) = \min \{ \tilde{t}_f, \beta_d(n_0), \beta_{\max}, \psi^{-1}(R/\alpha_{\max}) \} \quad (3.50)$$

where $\beta_d(n_0)$ represents the upper β bound resulting from a previous inspection, e.g. for the model described in Chapter 2,

$$\beta_d(n_0) = n_0 + (\tilde{t}_d - n_0)t/t_{1j} . \quad (3.51)$$

Note that \tilde{t}_f^* , defined by equation (2.54) corresponds to $\tilde{t}_f^*(R_{\min})$.

- (ii) $\alpha^*(n_0, \beta)$: Lower bound for α , given n_0 for the D_2 subspace.

$$\alpha^*(n_0, \beta) = \max \{ \alpha_{\min}, R_{\min}/\psi(\beta), R_p/\psi(\beta_1(n_0, \beta)) \} \quad (3.52)$$

where $\beta_1(n_0, \beta)$ defines the translation of the proof load boundary,

$$\beta_1(n_0, \beta) = n_0 + (\beta - n_0) t_{1j}/t. \quad (3.53)$$

α^* (equation 2.58) and α_f^* (2.70) correspond to $\alpha^*(n_0, \beta)$ and $\alpha^*(n_0, \tilde{t}_f)$ respectively.

(iii) $R^*(n_0, \beta)$: Lower bound for R given n_0 and β for the D_2 subspace,

$$R^*(n_0, \beta) = \alpha^*(n_0, \beta) \psi(\beta) \quad (3.54)$$

$$= \max \{ \alpha_{\min} \psi(\beta), R_{\min}, R_p \psi(\beta) / \psi(\beta_1(n_0, \beta)) \} \quad (3.55)$$

R^* (2.84), $R_f^*(n_0)$ (2.90) and $R_f^*(\tilde{t}_1)$ correspond to $R^*(n_0, \beta)$, $R^*(n_0, \tilde{t}_f)$ and $R^*(\tilde{t}_1, \tilde{t}_f)$ respectively.

(iv) $\alpha_2(\beta, R)$: Upper bound for α given β and n_0 for a domain bounded below by $R = \alpha \psi(\beta)$,

$$\alpha_2(\beta, R) = \min \{ \alpha_{\max}, R / \psi(\beta) \} \quad (3.56)$$

(v) α_v : Lower bound for α for the D_1 subspace of uncracked structures,

$$\alpha_v = \max \{ \alpha_{\min}, R_{\min}, R_p \} \quad (3.57)$$

(vi) $\beta_1(n_0)$ lower bound for β given n_0 , for D_2

$$\beta_1(n_0) = \max \{ n_0, \beta_{\min}, \tilde{t}_1 \}. \quad (3.58)$$

(vii) $\beta_{p,R}$: Lower limit for β for an integration along the line $R = \alpha\psi(\beta)$ in D_2 ,

$$\beta_{p,R} = \max\{\psi^{-1}(R/\alpha_{\min}), \beta'_{p,R}, \beta_1(n_0)\} \quad (3.59)$$

where $\beta'_{p,R}$ is the solution of

$$R = R_p\psi(\beta)/\psi(\beta_1(n_0, \beta)). \quad (3.60)$$

For convenient reference, these functions and their definitions are listed in Table 3.3.

$\tilde{t}_f^*(R)$	$\min\{\tilde{t}_f, \beta_d(n_0), \beta_{\max}, \psi^{-1}(R/\alpha_{\max})\}$
$\alpha^*(n_0, \beta)$	$\max\{\alpha_{\min}, R_{\min}/\psi(\beta), R_p/\psi(\beta_1(n_0, \beta))\}$
$R^*(n_0, \beta)$	$\max\{\alpha_{\min}\psi(\beta), R_{\min}, R_p\psi(\beta)/\psi(\beta_1(n_0, \beta))\}$
$\beta_{p,R}$	$\max\{\psi^{-1}(R/\alpha_{\min}), \beta'_{p,R}, \beta_1(n_0)\}$ $\beta'_{p,R}$ soln of $R = R_p\psi(\beta)/\psi(\beta_1(n_0, \beta))$
$\alpha_2(\beta, R)$	$\min\{\alpha_{\max}, R/\psi(\beta)\}$
α_v	$\max\{\alpha_{\min}, R_{\min}, R_p\}$
$\beta_1(n_0)$	$\max\{n_0, \beta_{\min}, \tilde{t}_1\}$

Table 3.3. Limit functions and their definitions.

3.3.3. Conditions for the existence of an integration along the line $R = \alpha\psi(\beta)$.

Some of the reliability functions involve integrations along lines (in α, β space for given n_0) of constant R . Conditions for the existence of such integrations can be used to define limits for the n_0 integration. The arguments presented here generalise the results expressed by equation (2.79) for an integration along the line $R = R_{\min}$.

Referring to Figure 3.14, it is clear that an integration along $R = \alpha\psi(\beta)$ will exist if,

$$\hat{t}_f^*(R) > \beta_{p,R}$$

$$\text{or } \min\{\hat{t}_f, \beta_d(n_0), \beta_{\max}, \psi^{-1}(R/\alpha_{\max})\} > \max\{\psi^{-1}(R/\alpha_{\min}), \beta_{p,R}, \beta_1(n_0)\} \\ \dots \quad (3.61)$$

Equation (3.61) can be interpreted as twelve separate inequalities. For a given set of model parameters, only one of these inequalities will represent the limiting condition for the existence of the integration.

Some of the inequalities in (3.61) are always satisfied by the default limit conditions. Others, however, impose limits on the ranges of n_0 and R . Taking the first term of the right hand side of (3.61), the set of conditions,

$$\psi^{-1}(R/\alpha_{\min}) < \min\{\hat{t}_f, \psi^{-1}(R/\alpha_{\max})\}$$

are satisfied by the default limits. The condition

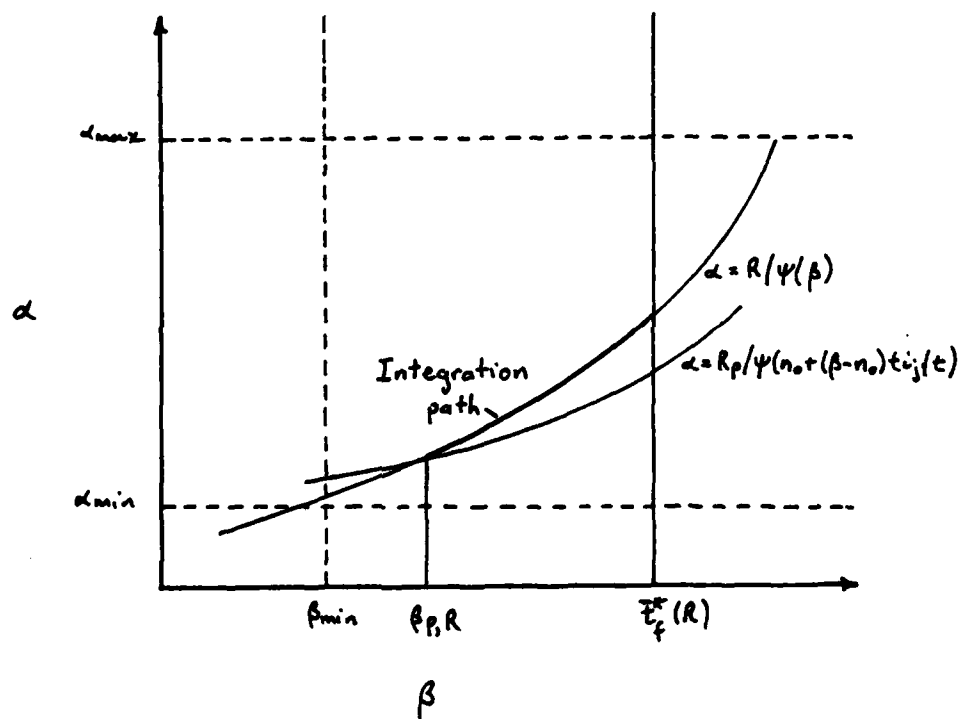


Figure 3.14. Definition of end points for an integration along the line $R = \alpha\psi(\beta)$.

$$\psi^{-1}(R/\alpha_{\min}) < \beta_{\max} \quad (3.62)$$

yields a lower limit for R. The condition,

$$\psi^{-1}(R/\alpha_{\min}) < \beta_d(n_0) \text{ is equivalent to}$$

$$n_0 < (t \cdot t_d - \psi^{-1}(R/\alpha_{\min}) t_1) / (t - t_1) \quad (3.63)$$

for $\beta_d(n_0)$ defined by (3.51).

Taking the second term of the right hand side of (3.61), three conditions can be combined,

$$\beta'_{p,R} < \beta' \quad (3.64)$$

$$\text{where} \quad \beta' = \min\{\tilde{t}_1, \beta_{\max}, \psi^{-1}(R/\alpha_{\max})\} \quad (3.65)$$

Equation (3.64) is equivalent to

$$R/\psi(\beta') > R_p/\psi(\beta_1(n_0, \beta')), \text{ which}$$

for $\beta_1(n_0, \beta)$ defined by (3.53) leads to

$$n_0 < (t \psi^{-1}(R_p \psi(\beta') / R) - \beta' t_1) / (t - t_1) \quad (3.66)$$

The condition

$$\beta_d(n_0) > \beta'_{p,R}$$

is equivalent to

$$n_0 < (t \cdot \tilde{t}_d - \psi^{-1}(R \psi(\tilde{t}_d) / R_p) t_1) / (t - t_1) \quad (3.67)$$

Taking the final term in the right hand side of (3.61) and remembering that

$$\beta_1(n_0) = \max\{n_0, \beta_{\min}, \tilde{t}_1\},$$

the nontrivial inequalities are,

$$\beta_d(n_0) > \beta_{\min} \quad , \quad (3.68)$$

$$\min \{ \psi^{-1}(R/\alpha_{\max}), \beta_{\max} \} > n_0 \quad (3.69)$$

$$\text{and } \psi^{-1}(R/\alpha_{\max}) > \beta_{\min} \quad . \quad (3.70)$$

Equation (3.68) together with (3.63) yields,

$$n_0 < (t \cdot \tilde{t}_d - \max \{ \psi^{-1}(R/\alpha_{\min}), \beta_{\min} \} t_{1j}) / (t - t_{1j}).$$

.... (3.71)

Now the inequality

$$\psi^{-1}(R_p \psi(\beta')/R) < \tilde{t}_d \quad \text{implies that,}$$

$$\psi^{-1}(R_p \tilde{t}_d / R_p) > \beta' \quad \text{so that (3.67) and}$$

(3.66) are mutually exclusive. Equation (3.71) is relevant

only when (3.67) may be. Equations (3.66), (3.67), (3.69) and (3.71)

therefore be combined,

$$n_0 < \min \left\{ \psi^{-1}(R/\alpha_{\max}), \beta_{\max}, \left[t \cdot \min \left\{ t_d, \psi^{-1}(R_p \psi(\beta')/R) \right\} \right. \right. \\ \left. \left. - \min \left\{ \beta', \max \left\{ \psi^{-1}(R/\alpha_{\min}), \psi^{-1}(R_p \tilde{t}_d / R_p), \beta_{\min} \right\} \right\} t_{1j} \right] \right\} \\ / (t - t_{1j}) \quad \text{....} \quad (3.72)$$

which equation can now be used as a generalized definition for

$f_{n_0}(R)$, which was previously defined by (2.79).

Equations (3.62) and (3.70) define limits for R ,

$$\alpha_{\min} \psi(\beta_{\max}) \leq R \leq \alpha_{\max} \psi(\beta_{\min}) \quad . \quad (3.73)$$

3.3.4. Conditions for the existence of integrations along the line $\beta = \tilde{t}_f$.

The calculation of the risk of fatigue life exhaustion, r_f , involves an integration along the line t_f . This integration exists if,

$$\tilde{t}_f^*(R_{\min}) \geq \tilde{t}_f, \quad (3.74)$$

which is equivalent to the three conditions,

$$\beta_{\max} \geq \tilde{t}_f, \quad (3.75)$$

$$\psi^{-1}(R_{\min}/\alpha_{\max}) \geq \tilde{t}_f, \quad (3.76)$$

and
$$\beta_d(n_0) \geq \tilde{t}_f. \quad (3.77)$$

Condition (3.76) is always met as a consequence of the default limit, (3.46). Condition (3.77) yields an upper limit for n_0 ; for $\beta_d(n_0)$ defined by (3.51),

$$n_0 \leq (\tilde{t}_d t - \tilde{t}_f t_{1j}) / (t - t_{1j}) \quad (3.78)$$

$$\leq i_{n_0} \quad \text{as defined by (2.80).}$$

3.4. Specification for the Most General Model

Having detailed the effects of the characteristics and limitations of the input data, it is now possible to formulate complete specifications for the reliability functions evaluated by NERF and the remainder of this Chapter is devoted to this task.

The discussions to this point have been primarily concerned with the mathematical development of the various reliability functions and the notation has been generally optimized to that end. Throughout the remainder of this document, the reliability expressions will resemble, as closely as possible, the forms actually used to generate the computer code. (One example is the fact that the code performs the outermost integration over initial crack length (a_0) rather than initial age (n_0).)

Many of the expressions presented in the remaining sections of this Chapter will be stated without necessarily presenting the full details of their development. It is assumed that the arguments presented in the previous sections are sufficient for the reader to derive the same results without detailed guidance.

The model classification system described in Section 3.1 is used: the most general model is the class allowing variations in both \underline{X}_1 and \underline{X}_3 and includes the three groupings based on the status of the initial crack length random variable, \underline{X}_2 .

The specification for this model contains two parts; a specification for the integration over crack length for each reliability function and a specification for the integrands. The first part is general and applies to all the classes of models evaluated by NERF. The second part pertains only to the most general class.

3.4.1. Specifications for the reliability functions as integrations over initial crack length

Each reliability function can be written in the form of equation (3.1) as an integration over initial life or, alternatively, as an integration over initial crack length. The former choice was preferred while describing the theoretical development. The computer program actually retains initial crack length (X_2 or a_0) as the variable of integration so that (3.1) is replaced by,

$$f(t) = \int_{a(n_1)}^{a(n_2)} p_{a_0}(a_0) f(t, a^{-1}(a_0)) da_0 \quad . \quad (3.79)$$

Note that the integrand is still expressed as a function of n_0 . The numerical evaluation of the reliability functions is, in fact, based on a hybrid notation whereby a_0 is used for the outermost integration and n_0 is used for all other representations of the second random variable in the model. (The reasons for using the hybrid notation are that it is numerically more efficient, bearing in mind that the function $a(\beta)$ is defined as a series of ordered pairs, to evaluate $p_{a_0}(a_0)$ directly rather than $p_{n_0}(n_0) = p_{a_0}(a(n_0)) da/dn_0$ and that the evaluation of the limits of integration and other logical processes associated with the inner levels of integration are more efficiently expressed in terms of n_0 rather than a_0 .)

The following specification for the initial crack length integrations for the reliability functions evaluated by NERF follow, broadly, equations (3.2) to (3.17). The major differences

are that the equations presented here are recast in the form of (3.79) and that some functions represent combinations of functions that were previously treated as being separate. Note also that the probability of survival is not evaluated as an integral expression but is, instead, derived from the probability of failure via equation (2.5).

The reliability functions evaluated by NERF are:

(i) Probability of failure

$$P_F(t) = \int_{a(n_1)}^{a(n_2)} P_F(t, n_0) p_{a_0}(a_0) da_0 + \bar{P}_{a_0}(a(n_2)) ;$$

(from (3.5)). (3.80)

(ii) Probability of survival

$$P_S(t) = 1 - P_F(t) \quad (3.81)$$

(iii) Probability of detection

$$P_{det}(t) = \int_{a(n_{d1})}^{a(n_2)} P_{det}(t, n_0) p_{a_0}(a_0) da_0 ; \quad (3.82)$$

(from (3.6)).

(iv) Virgin risk rate

$$r_v(t) = \delta(n_0, 0) P_{FV}(t) / P_S(t); \quad (3.83)$$

(v) Risk of static failure by fatigue

$$r_s(t) = \int_{a(n_1)}^{a(n_2)} P_{fs}(t, n_0) p_{a_0}(a_0) da_0 / P_s(t); \quad (3.84)$$

(vi) Risk of fatigue fracture

$$r_f(t) = \int_{a(n_{f1})}^{a(n_{f2})} [P_{ff1}(t, n_0) + P_{ff2}(t, n_0)] p_{a_0}(a_0) da_0 / P_s(t); \quad (3.85)$$

.....

(vii) Total risk

$$r(t) = r_v(t) + r_f(t) + r_s(t); \quad (3.86)$$

(viii) Density for strength

$$P_R(R | F > t) = \int_{a(n_{R1})}^{a(n_{R2})} f_{R\lambda}(t, n_0) p_{a_0}(a_0) da_0 / P_s(t); \quad (3.87)$$

(from (3.14))

(ix) Failure density for strength

$$P_R(R | t) = \int_{a(n_{R1})}^{a(n_{R2})} [F_R(R, t, n_0) r_2(R) + f(R, R_{min}) P_{ff1}(t, n_0) + f_{R,f2}(R, t, n_0)] p_{a_0}(a_0) da_0 / (P_s(t) r(t)). \quad (3.88)$$

....

(from (3.17), using (2.9)).

The functions $P_F(t, n_0)$, $P_{det}(t, n_0)$, $P_{fv}(t)$, $P_{fs}(t, n_0)$, $P_{ff1}(t, n_0)$, $P_{ff2}(t, n_0)$, $f_R(R, t, n_0)$ and $f_{R, f2}(R, t, n_0)$ were defined in Section 3.1. These functions, together with the limits n_1 , n_2 , n_{f1} , n_{f2} , n_{d1} , n_{R1} and n_{R2} may change between classes of model and are therefore specified separately for each class.

Note that, although the caution was given in Section 3.1 that, in general, the limits n_1 and n_2 may vary between reliability functions, all the models described herein are such that all the variation can be accommodated by the addition of the five extra limits n_{d1} , n_{f1} , n_{f2} , n_{R1} and n_{R2} . In equations (3.80) to (3.88) n_1 and n_2 remain the same for a given class.

(Note that expressions for the extra limits will be given later in this Chapter.)

3.4.2. Specification for the integrand functions for the most general model

The integrand functions for the 'full model' which accounts for variations in both X_1 and X_2 may now be specified. Each expression can be derived by examination of the appropriate equations in Chapter 2 and including the correct limit functions from Table 3.3. For this class, the following functions have a special form;

$$p_{\beta}(\beta) = p_{X_1}(t/(\beta - n_0))t/(\beta - n_0)^2, \quad (3.89)$$

$$H(\alpha, \beta, n_0, t) = \exp\left\{-t/(\beta - n_0)\left[\delta(n_0, 0)r_1(\alpha)\bar{t}_1 + \int_{\max\{\bar{t}_1, n_0\}}^{\beta} r_2(\alpha, \beta')d\beta'\right]\right\}$$

... (3.90)

$$\beta_d(n_0) = n_0 + (\bar{t}_d - n_0)t/t_{1j} \quad (3.91)$$

$$\beta_1(n_0, \beta) = n_0 + (\beta - n_0)t_{1j}/t \quad (3.92)$$

The integrand functions are;

$$P_F(t, n_0) = \int_{\beta_1(n_0)}^{\bar{t}_F^*(R_{min})} p_{\beta}(\beta) \int_{\alpha^*(n_0, \beta)}^{\alpha_2(\beta, R_{max})} (1 - H(\alpha, \beta, n_0, t)) p_{\alpha}(\alpha) d\alpha d\beta$$

$$+ \int_{\beta_1(n_0)}^{\bar{t}_F^*(R_{min})} p_{\beta}(\beta) p_{\alpha}(\alpha^*(n_0, \beta)) d\beta + \bar{p}_{\beta}(\bar{t}_F^*(R_{min}))$$

$$+ \delta(n_0, 0) \left[\int_{\alpha_v}^{\alpha_2(0, R_{max})} p_{\alpha}(\alpha) (1 - \exp\{-r_1(\alpha)t\}) d\alpha + p_{\alpha}(\alpha_v) \right] p_{\beta}(\bar{t}_1)$$

..... (3.93)

$$\begin{aligned}
P_{det}(t, n_0) = & \int_{\beta_1(n_0)}^{\tilde{t}_d} p_\beta(\beta) \int_{\alpha^*(n_0, \beta)}^{\alpha_2(\beta, R_p)} p_\alpha(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \\
& + \int_{\tilde{t}_d}^{\tilde{t}_f^*(R_{min})} p_\beta(\beta) \int_{\alpha^*(n_0, \beta)}^{\alpha_{max}} p_\alpha(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta \\
& + \delta(n_0, 0) \delta(t, t_1) p_\beta(\tilde{t}_1) \int_{\alpha_{min}}^{\alpha_v} p_\alpha(\alpha) \exp\{-r_1(\alpha)t\} d\alpha \quad (3.94)
\end{aligned}$$

$$P_{fv}(t) = \int_{\alpha_v}^{\alpha_2(0, R_{max})} p_\alpha(\alpha) r_1(\alpha) \exp\{-r_1(\alpha)t\} d\alpha p_\beta(\tilde{t}_1), \quad (3.95)$$

$$\begin{aligned}
P_{fs}(t, n_0) = & \int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} p_\beta(\beta) \int_{\alpha^*(n_0, \beta)}^{\alpha_2(\beta, R_{max})} r_2(\alpha, \beta) H(\alpha, \beta, n_0, t) p_\alpha(\alpha) d\alpha d\beta, \\
& \dots \quad (3.96)
\end{aligned}$$

$$P_{ff1}(t, n_0) = \int_{R_{min}/\psi(\beta_p, R_{min})}^{\alpha_2(\tilde{t}_f^*(R_{min}), R_{min})} p_\alpha(\alpha) H(\alpha, \beta^*, n_0, t) p_\beta(\beta^*) \frac{(\beta^* - n_0)}{t} d\alpha, \quad (3.97)$$

$$P_{ff2}(t, n_0) = \delta(\tilde{t}_f, \tilde{t}_f^*(R_{min})) p_\beta(\tilde{t}_f) \frac{(\tilde{t}_f - n_0)}{t} \int_{\alpha^*(n_0, \tilde{t}_f)}^{\alpha_{max}} p_\alpha(\alpha) H(\alpha, \tilde{t}_f, n_0, t) d\alpha, \quad (3.98)$$

$$\begin{aligned}
f_R(R, t, n_0) = & \int_{\beta_p, R}^{\tilde{t}_f^*(R)} \frac{p_\beta(\beta) p_\alpha(R/\psi(\beta)) H(R/\psi(\beta), \beta, n_0, t) d\beta}{r(\beta)} \\
& + \delta(n_0, 0) R_s(R - R_p) p_\alpha(R) \exp\{-r_1(R)t\} p_\beta(\tilde{t}_1), \quad (3.99)
\end{aligned}$$

$$f_{R,f2}(R,t,n_0) = \frac{H_S(R - R^*(\tilde{t}_1, \tilde{t}_f)) p_\theta(\tilde{t}_f)(\tilde{t}_f - n_0) p_\lambda(R/\psi(\tilde{t}_f) \delta(\tilde{t}_f, \tilde{t}_f^*(R_{\min}))}{t \psi(\tilde{t}_f)} \times H(R/\psi(\tilde{t}_f), \tilde{t}_f, n_0, t) \quad (3.100)$$

where β^* in (3.97) is equivalent to $\psi^{-1}(R_{\min}/\omega)$.

The terms appearing in the integration limits for n_0 in equations (3.80) to (3.88) are;

$$n_1 = n_{d1} = n_{0,\min}, \text{ (as defined by (3.48))}, \quad (3.101)$$

$$n_2 = \min \left\{ n_{0,\max}, (t \cdot \tilde{t}_d - \beta_{\min} t_{1j}) / (t - t_{1j}) \right\} \quad (3.102)$$

(where $n_{0,\max}$ is defined by (3.49))

$$n_{f1} = n_{0,\min} \quad (3.103)$$

$$n_{f2} = \min \left\{ n_{0,\max}, \max \{ f_{n_0}, f_{n_0}(R_{\min}) \} \right\} \quad (3.104)$$

$$n_{R1} = n_{0,\min} \quad (3.105)$$

$$n_{R2} = \min \{ f_{n_0}(R), n_{0,\max} \} \quad (3.106)$$

where $f_{n_0}(R)$ and f_{n_0} are defined by (3.72) and (3.78) respectively.

3.5. Derivations and Specifications for Simpler Models

The three remaining classes of models have variations in at least one of the random variables \underline{X}_1 and \underline{X}_3 ignored and may therefore be regarded as simplifications of the full model specified in the previous Section. However, the derivation of the appropriate integrand functions and limits is not necessarily a straightforward reduction of equations (3.89) to (3.106). For each class there are some functions for which a more satisfactory approach is to derive the function using the general methodology described in Chapter 2 and the assumptions appropriate for the reduced model.

For this reason, the specifications for the remaining three classes of models are presented separately and that for a given class is preceded by a brief description of the derivations of those functions which are not obvious reductions of equations (3.89) to (3.106).

3.5.1. Constant relative strength

If \bar{X}_3 is held constant, the virgin strength α will, via equation (2.29), also be constant. The convention taken here is to set $\bar{X}_3 = 1$ so that \bar{R}_0 is the constant value of virgin strength. The reliability functions for this class can generally be derived from the more complex forms by setting,

$$\alpha_{\max} = \alpha_{\min} = \bar{R}_0 \quad (3.107)$$

and ignoring the integration over α . For example, the expression for the probability of survival for the full initial crack model of this class is, (from (2.57))

$$P_S(t) = \int_{n_1}^{n_2} P_{n_0}(n_0) \int_{\beta_1(n_0)}^{\bar{t}_f^*(R_{\min})} p_{\beta}(\beta) H(R_0, \beta, n_0, t) d\beta dn_0 \quad (3.108)$$

An important consequence of the default limiting relationships (3.44) and (3.46) is that

$$\bar{t}_f = \psi^{-1}(R_{\min}/\bar{R}_0) \quad (3.109)$$

Failures resulting from fatigue life limiting are the same as those resulting from the strength being reduced to R_{\min} . Which of the two identical terms, $(p_{ff1}(t, n_0)$ or $P_{ff2}(t, n_0)$), is taken for $r_f(t)$ is arbitrary and it is convenient for the purpose of representing the failure density for strength to associate $r_f(t)$ with failures at $R = R_{\min}$.

For this class of models, proof load inspections are identical in effect to crack length inspections with

$$\bar{t}_d = \psi^{-1}(R_p/\bar{R}_0) \quad (3.110)$$

The derivation of the density for strength requires careful reconsideration and can be made by transforming the β integration in (3.108) to one over R , viz.,

$$\begin{aligned}
 P_S(t) &= \int_{n_1}^{n_2} P_{n_0}(n_0) \int_{\tilde{R}_0 \psi(\tilde{t}_f(R_{\min}))}^{\tilde{R}_0 \psi(\beta_1(n_0))} \frac{P_{\beta}(\psi^{-1}(R/\tilde{R}_0)) H(\tilde{R}_0, \psi^{-1}(R/\tilde{R}_0), n_0, t) dR dn_0}{\tilde{R}_0 |\psi'(\psi^{-1}(R/\tilde{R}_0))|} \\
 &= \int_{R_{\min}}^{\tilde{R}_0} \int_{n_1}^{f_{n_0}(R)} \frac{P_{n_0}(n_0) P_{\beta}(\psi^{-1}(R/\tilde{R}_0)) H(\tilde{R}_0, \psi^{-1}(R/\tilde{R}_0), n_0, t) dn_0 dR}{\tilde{R}_0 |\psi'(\psi^{-1}(R/\tilde{R}_0))|} .
 \end{aligned}
 \tag{3.111}$$

(where the prime denotes differentiation by the argument of ψ .) Equation (3.111) leads to the first term in $f_R(R, t, n_0)$ listed below (3.122). The second term can be derived by reduction from (3.99).

The conditions for the existence of the integrand in (3.111) are,

$$\beta_1(n_0) < \psi^{-1}(R/\tilde{R}_0) < \tilde{t}_f^*(R_{\min}) \tag{3.112}$$

where $\tilde{t}_f^*(R_{\min})$ is, from (3.50), given by,

$$\tilde{t}_f^*(R_{\min}) = \min \{ \tilde{t}_f, \beta_d(n_0), \beta_{\max} \} . \tag{3.113}$$

Equation (3.112) can be used to yield,

$$f_{n_0}(R) = \min \{ \psi^{-1}(R/\tilde{R}_0), (\tilde{t}_d t - \psi^{-1}(R/\tilde{R}_0) t_{1j}) / (t - t_{1j}) \} \tag{3.114}$$

and

$$\tilde{R}_0 \psi(\min \{ \beta_{\max}, \tilde{t}_f \}) < R < \tilde{R}_0 \psi(\max \{ \beta_{\min}, \tilde{t}_i \}) \tag{3.115}$$

Note that a consequence of equation (3.109) is that $f_{n_0} = f_{n_0}(R_{min})$. Provided (3.114) is used for $f_{n_0}(R)$, equations (3.101) to (3.106) are valid for the a_0 integration limits for this class.

The a_0 integrand functions are;

$$P_F(t, n_0) = \int_{\beta_1(n_0)}^{\tilde{t}_F^*(R_{min})} p_F(\beta) [1 - H(\tilde{R}_0, \beta, n_0, t)] d\beta + p_F(\tilde{t}_F^*(R_{min})) + \delta(n_0, 0) [1 - \exp\{-r_1(\tilde{R}_0)t\}] p_F(\tilde{t}_1), \quad (3.116)$$

$$P_{det}(t, n_0) = \int_{\tilde{t}_d}^{\tilde{t}_F^*(R_{min})} p_F(\beta) H(\tilde{R}_0, \beta, n_0, t) d\beta, \quad (3.117)$$

$$P_{fv}(t) = r_1(\tilde{R}_0) \exp\{-r_1(\tilde{R}_0)t\} p_F(\tilde{t}_1), \quad (3.118)$$

$$P_{fs}(t, n_0) = \int_{\beta_1(n_0)}^{\tilde{t}_F^*(R_{min})} p_F(\beta) r_2(R_0, \beta) H(R_0, \beta, n_0, t) d\beta, \quad (3.119)$$

$$\dot{P}_{ff}(t, n_0) = \frac{p_F(\tilde{t}_F)(\tilde{t}_F - a_0) H(\tilde{R}_0, \tilde{t}_F, n_0, t) \delta(\tilde{t}_F, \tilde{t}_F^*(R_{min}))}{t}, \quad (3.120)$$

$$p_{ff2}(t, n_0) = 0, \quad (3.121)$$

$$f_R(t, n_0) = \frac{p_R(\psi^{-1}(R/\tilde{R}_0))H(R_0, \psi^{-1}(R/\tilde{R}_0), n_0, t)}{\tilde{R}_0 |\psi'(\psi^{-1}(R/\tilde{R}_0))|}, \quad (3.122)$$

$$+ \delta(n_0, 0) \delta(R, \tilde{R}_0) \exp\{-r_1(\tilde{R}_0) t\} p_R(\tilde{t}_1)$$

$$f_{R, f2}(t, n_0) = 0. \quad (3.123)$$

3.5.2. Constant relative fatigue life

If X_1 is held constant, there is no variation in relative fatigue life. If $X_1 = 1$, the age of any structure is affected only by the random variable n_0 and time, i.e.,

$$\beta = t + n_0 \quad (3.124)$$

$$\text{and } \beta_{\min} = t + n_{0,\min}, \quad \beta_{\max} = t + n_{0,\max} \quad (3.125)$$

Since the derivation of the reliability functions for the two classes already described were based on a transformation from X_1 to β , the reliability functions for this class cannot be generated by a simple reduction from more general equations. The limit equations can be generated, however, provided suitable expressions for $\beta_d(n_0)$ and $\beta_1(n_0, \beta)$ are found. From (2.51) with $x_1 = 1$,

$$\beta_d(n_0) = \bar{t}_d - t_{1j} + t \quad (3.126)$$

and by following the proof load arguments, (from (3.53))

$$\beta_1(n_0, \beta) = \beta - t + t_{1j} \quad (3.127)$$

$$= n_0 + t_{1j} \quad (3.128)$$

The integration domain for the D_2 subspace can be delineated in (n_0, α) or (β, α) space as shown in Figure 3.15 which replaces Figure 3.13 for this class.

The probability of survival for the full initial crack model for this class is, (from (2.59))

$$P_S(t) = \int_{n_1}^{n_2} p_{n_0}(n_0) \int_{\alpha^*(n_0, t+n_0)}^{\alpha_{\max}} R_{\alpha}(\alpha) H(\alpha, t+n_0, n_0, t) d\alpha dn_0 \quad (3.129)$$

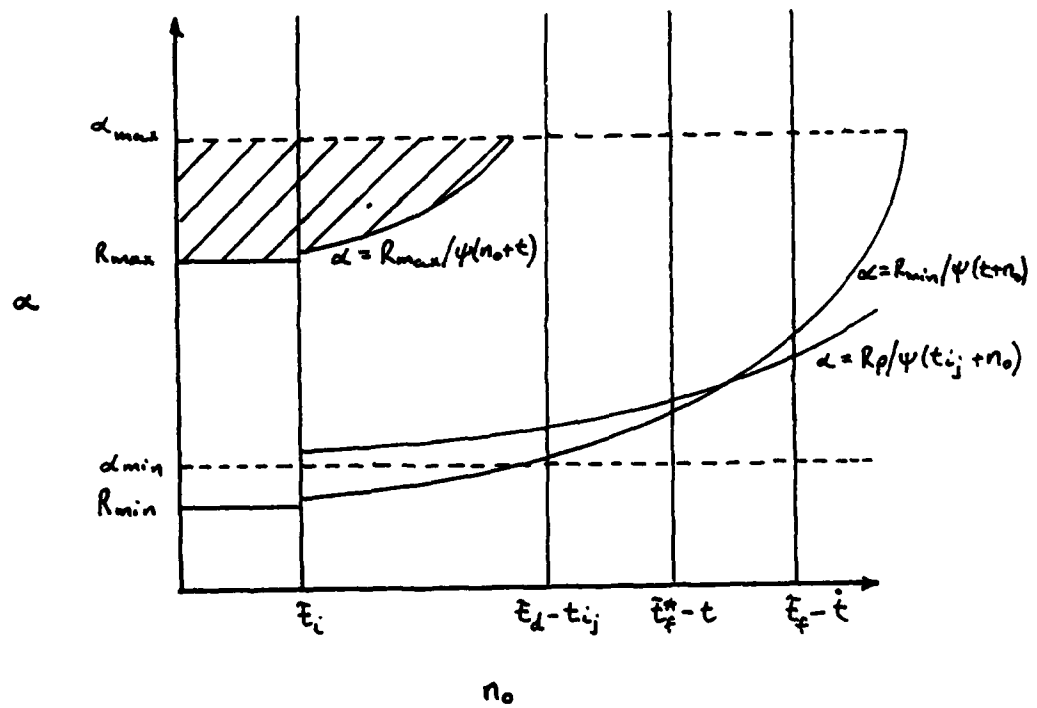


Figure 3.15. Typical juxtaposition of limits for the case of constant relative fatigue life. Compare with Figure 3.13.

Note that the integrand for n_0 in (3.129) exists only when,

$$n_0 + t < \tilde{t}_f^*(R_{\min}) \quad (3.130)$$

so that n_2 becomes,

$$n_2 = \min\{n_{0,\max}, \tilde{t}_f^*(R_{\min}) - t\} \quad (3.131)$$

which replaces (3.102).

Following the differentiation with respect to time described in Section 2.2.7, and noting that $\frac{\partial \beta}{\partial t} = 1$, the probability of failure for the full initial crack model for this class is,

$$\begin{aligned} P_F(t) = & \int_{n_1}^{n_2} p_{n_0}(n_0) \left\{ \begin{array}{l} \alpha_2(n_0+t, R_{\max}) \\ r_2(\alpha, n_0+t) H(\alpha, n_0+t, n_0, t) p_\alpha(\alpha) d\alpha dn_0 \\ \alpha^*(n_0, n_0+t) \end{array} \right. \\ & + \int_{R_{\min}/\psi(\beta_p, R_{\min})}^{R_{\min}/\psi(\tilde{t}_f^*(R_{\min}))} p_{n_0}(\beta^* - t) p_\alpha(\alpha) H(\alpha, \beta^*, \beta^* - t, t) d\alpha \\ & + \delta(\tilde{t}_f^*(R_{\min}), \tilde{t}_f) p_{n_0}(\tilde{t}_f - t) \int_{\alpha^*(\tilde{t}_f - t, \tilde{t}_f)}^{\alpha_{\max}} p_\alpha(\alpha) H(\alpha, \tilde{t}_f, \tilde{t}_f - t, t) d\alpha \\ & \dots \quad (3.132) \end{aligned}$$

The second term of (3.132) which represents failures at $R = R_{\min}$ can be transformed to an integration over n_0 . Denoting this term by $r_{f1}(t)$,

$$r_{f1}(t) = \int_{\beta_{p,R}^{-t}}^{n_2} p_{n_0}(n_0) p_{\alpha}(R_{\min}/\psi(n_0+t)) H(R_{\min}/\psi(n_0+t), n_0+t, n_0, t) \times \frac{R_{\min} \psi'(n_0+t)}{\psi(n_0+t)^2} dn_0 \quad (3.133)$$

which leads to the expression for $p_{f1}(t, n_0)$ listed below.

The expression for $p_R(R|F>t)$ can be generated by transforming the inner integration in (3.129) to one over R and then reversing the order of integration. Using $R = \alpha\psi(n_0+t)$,

$$\begin{aligned} p_R(t) &= \int_{n_1}^{n_2} p_{n_0}(n_0) \int_{R^*(n_0, n_0+t)}^{\alpha_{\max} \psi(n_0+t)} \frac{p_{\alpha}(R/\psi(n_0+t)) H(\alpha, n_0+t, n_0, t)}{\psi(n_0+t)} dR dn_0 \\ &= \int_{\alpha_{\min} \psi(\tilde{t}_1^*(R_{\min}))}^{\alpha_{\max} \psi(\tilde{t}_1+t)} \int_{\beta_{p,R}^{-t}}^{\tilde{t}_1^*(R) - t} \frac{p_{n_0}(n_0) p_{\alpha}(R/\psi(n_0+t)) H(\alpha, n_0+t, n_0, t)}{\psi(t+n_0)} dn_0 dR \\ &\dots\dots (3.134) \end{aligned}$$

which can be used to generate the expression for $f_R(t, n_0)$ listed below (the second term can be deduced by reduction from equation (3.99)).

The a_0 integrand functions are;

$$\begin{aligned}
 P_F(t, n_0) &= \int_{\alpha^*(n_0, n_0+t)}^{\alpha_2(n_0+t, R_{\max})} p_\alpha(\alpha) [1 - H(\alpha, t+n_0, n_0, t)] d\alpha \\
 &\quad + p_\alpha(\alpha^*(n_0, n_0+t)) \quad \tilde{t}_1 \leq n_0+t < \tilde{t}_2 \\
 \text{or } \delta(n_0, 0) &\int_{\alpha_v}^{\alpha_2(0, R_{\max})} (1 - \exp\{-r_1(\alpha)t\}) p_\alpha(\alpha) d\alpha; n_0+t < \tilde{t}_1 \\
 &\quad + p_\alpha(\alpha_v) \dots \quad (3.135)
 \end{aligned}$$

$$\begin{aligned}
 P_{det}(t, n_0) &= \int_{\alpha^*(n_0, n_0+t)}^{\alpha_2(n_0+t, R_p)} p_\alpha(\alpha) H(\alpha, t+n_0, n_0, t) d\alpha \quad \tilde{t}_1 \leq n_0+t < \tilde{t}_2 \\
 \text{or } \delta(n_0, 0) \delta(t, t_1) &\int_{\alpha_{\min}}^{\alpha_v} p_\alpha(\alpha) \exp\{-r_1(\alpha)t\} d\alpha \quad t+n_0 < \tilde{t}_1 \\
 &\dots \quad (3.136)
 \end{aligned}$$

$$\begin{aligned}
 P_{fv}(t) &= \int_{\alpha_v}^{\alpha_2(0, R_{\max})} \exp\{-r_1(\alpha)t\} r_1(\alpha) p_\alpha(\alpha) d\alpha \quad n_0+t < \tilde{t}_1 \\
 &\dots \quad (3.137)
 \end{aligned}$$

$$\begin{aligned}
 P_{fs}(t, n_0) &= \int_{\alpha^*(n_0, n_0+t)}^{\alpha_2(n_0+t, R_{\max})} r_2(\alpha, n_0+t) H(\alpha, n_0+t, n_0, t) p_\alpha(\alpha) d\alpha \quad \tilde{t}_1 \leq n_0+t < \tilde{t}_2 \\
 &\dots \quad (3.138)
 \end{aligned}$$

$$p_{ff1}(t, n_0) = \frac{p_\alpha(R_{\min}/\psi(n_0+t))H(R_{\min}/\psi(n_0+t), n_0+t, n_0, t)R_{\min}\psi'(n_0+t)}{\psi(n_0+t)^2} \quad \bar{t}_1 \leq n_0+t < \bar{t}_f, \quad \dots \quad (3.139)$$

$$p_{ff2}(t, n_0) = \delta(\bar{t}_f, \bar{t}_f^*(R_{\min}))\delta(\bar{t}_f-t, n_0) \int_{\alpha^*(\bar{t}_f-t, \bar{t}_f)}^{\alpha_{\max}} p_\alpha(\alpha)H(\alpha, \bar{t}_f, \bar{t}_f-t, t)d\alpha, \quad \dots \quad (3.140)$$

$$f_R(t, n_0) = p_\alpha(R/\psi(n_0+t))H(R/\psi(n_0+t), n_0+t, n_0, t)/\psi(n_0+t), \bar{t}_1 \leq n_0+t < \bar{t}_f, \quad \dots \quad (3.141)$$

$$\text{or } \delta(n_0, 0)H_S(R-R_p)p_\alpha(R)\exp\{-r_1(R)t\} \quad n_0+t < \bar{t}_1, \quad \dots \quad (3.142)$$

$$f_{R,f2}(n_0, t) = H_S(R - R^*(\bar{t}_1, \bar{t}_f)\delta(\bar{t}_f, \bar{t}_f^*(R_{\min}))\delta(\bar{t}_f-t, n_0) \\ p_\alpha(R/\psi(\bar{t}_f))H(R/\psi(\bar{t}_f), \bar{t}_f, \bar{t}_f-t, t)/\psi(\bar{t}_f) \quad (3.143)$$

The terms appearing in the integration limits for a_0 are;

$$n_1 = n_{0,\min} \quad (3.144)$$

$$n_2 = \min\{n_{0,\max}, \bar{t}_f^*(R_{\min})-t\} \quad (3.145)$$

$$n_{f1} = \max\{n_{0,\min}, \beta'_{p,R_{\min}}-t\} \quad (3.146)$$

$$n_{r2} = n_2, \quad (3.147)$$

$$n_{d1} = \max\{n_1, \bar{t}_d - t\} \quad (3.148)$$

$$n_{R1} = \max\{n_0, \min\{\beta'_{p,R} - t\}, \quad (3.149)$$

and

$$n_{R2} = n_2. \quad (3.150)$$

3.5.3. Constant relative strength and relative fatigue life

The final class represents the combination of the simplifications of the two previous classes. X_1 and X_3 are held constant so that,

$$\beta = t + n_0, \quad (3.151)$$

$$\alpha = \tilde{R}_0, \quad (3.152)$$

$$\alpha_{\min} = \alpha_{\max} = \tilde{R}_0, \quad (3.153)$$

and $\beta_{\min} = n_{0,\min} + t \quad \beta_{\max} = n_{0,\max} + t \quad (3.154)$

The expression for the probability of survival for the full initial crack model for this class is, (from (2.57))

$$P_S(t) = \int_{n_{0,\min}}^{\tilde{t}_f^*(R_{\min}) - t} p_{n_0}(n_0) H(\tilde{R}_0, t + n_0, n_0, t) dn_0 \quad (3.155)$$

and, as was the case for the constant relative strength class, proof load and crack length inspections are identical. Differentiating (3.153) with respect to time leads to,

$$\begin{aligned} P_S(t) = & \int_{n_{0,\min}}^{\tilde{t}_f^*(R_{\min}) - t} r_2(\tilde{R}_0, t + n_0) p_{n_0}(n_0) H(\tilde{R}_0, t + n_0, n_0, t) dn_0 \\ & + \delta(\tilde{t}_f, \tilde{t}_f^*(R_{\min})) \delta(\tilde{t}_f - t, n_0) p_{n_0}(\tilde{t}_f - t) H(R_0, \tilde{t}_f, \tilde{t}_f - t, t) . \\ & \dots \quad (3.156) \end{aligned}$$

There is only one r_f term.

The derivation of $p_R(R | \underline{F} > t)$ is made by transforming the n_0 integration in (3.153) to one over R in a similar manner to that used to derive (3.111). The result of the transformation is,

$$p_{\underline{S}}(t) = \int_{R_0 \psi(t_{f_{\min}}^*)}^{R_0 \psi(n_{0,\min})} \frac{p_{n_0}(\psi^{-1}(R/\tilde{R}_0) - t) H(\tilde{R}_0, \psi^{-1}(R/\tilde{R}_0), \psi^{-1}(R/\tilde{R}_0) - t, t) dR}{\tilde{R}_0 |\psi'(\psi^{-1}(R/\tilde{R}_0))|} \quad \dots (3.157)$$

which yields the expression for $f_R(t, n_0)$ listed below. Note that, given R , $f_R(t, n_0)$ exists only for a given value of n_0 , i.e.,

$$n_0 = \psi^{-1}(R/\tilde{R}_0) - t \quad (3.158)$$

The a_0 integrand functions are;

$$\begin{aligned} p_{\underline{F}}(t, n_0) &= 1 - H(\tilde{R}_0, n_0 + t, n_0, t) & \tilde{t}_1 \leq n_0 + t < \tilde{t}_f \\ &\text{or } 1 - \exp\{-r_1(R_0)t\} & n_0 + t < \tilde{t}_1 \\ &\dots & (3.159) \end{aligned}$$

$$p_{det}(t, n_0) = H(\tilde{R}_0, n_0 + t, n_0, t) \quad \tilde{t}_d \leq n_0 + t < \tilde{t}_f \quad (3.160)$$

$$p_{fv}(t) = \delta(n_0, 0) r_1(R_0) \exp\{-r_1(R_0)t\} \quad n_0 + t < \tilde{t}_1 \quad (3.161)$$

$$\begin{aligned} p_{fs}(t, n_0) &= r_2(R_0, t + n_0) H(R_0, t + n_0, n_0, t) & \tilde{t}_1 \leq t + n_0 < \tilde{t}_f \\ &\dots & (3.162) \end{aligned}$$

$$P_{ff1}(t, n_0) = \delta(\tilde{t}_f^*(R_{\min}), \tilde{t}_f) \delta(\tilde{t}_f - t, n_0) H(\tilde{R}_0, \tilde{t}_f, \tilde{t}_f - t, t) \quad (3.163)$$

$$P_{ff2}(t, n_0) = 0 \quad (3.164)$$

$$f_R(t, n_0) = \delta(n_0, \psi^{-1}(R/\tilde{R}_0) - t) \frac{H(R_0, \psi^{-1}(R/\tilde{R}_0), \psi^{-1}(R/\tilde{R}_0) - t, t)}{\tilde{R}_0 |\psi'(\psi^{-1}(R/\tilde{R}_0))|} \quad (3.165)$$

$t + n_0 < \tilde{t}_1$

$$f_{R,f2}(R, t, n_0) = 0 \quad (3.166)$$

The terms appearing in the integration limits for n_0 are;

$$n_1 = n_{0,\min} \quad (3.167)$$

$$n_2 = \min\{n_{0,\max}, \tilde{t}_f^*(R_{\min}) - t\} \quad (3.168)$$

$$n_{f1} = \tilde{t}_f - t \quad (3.169)$$

$$n_{f2} = \tilde{t}_f - t \quad (3.170)$$

$$n_{d1} = \tilde{t}_d - t \quad (3.171)$$

$$n_{R1} = \psi^{-1}(R/\tilde{R}_0) - t \quad (3.172)$$

and $n_{R2} = \psi^{-1}(R/\tilde{R}_0) - t \quad (3.173)$

Note that coincident limits in (3.172) and (3.173) imply a point evaluation of the integrand rather than a zero integration.

3.6. The Construction of a Time Sequence

The specifications detailed in the previous Sections are complete in that they define the reliability functions so that they may, in principle, be evaluated at any value of time. Moreover, each function is defined in such a way that knowledge of the prior history of the function is not required. Even the inspection procedures can be correctly incorporated if the time of only the latest inspection is defined.

However, the evaluation of the reliability functions at a single instant of time usually provides little information regarding the behaviour of the model. A complete time history is essential, particularly if several inspections are involved. The development of a time sequence of reliability functions is thus a significant feature of the NERF computer program.

A time sequence is normally comprised of three sub-sequences defined by the user as described below.

(1) Evaluation times

A sequence of N evaluation times, t_n , form the basis of the time sequence. These values may be specified in any way, but are evaluated in ascending order.

(ii) Inspection times

J inspection times t_{ij} can be included in the sequence. There are several ways in which these times can be either specified as input data, or calculated by NERF. Section 3.6.2. below details the options available.

(iii) Strength distribution times

A sequence of times at which the density of strength and the failure density of strength are computed (for a series of values of R) can also be selected. For a given time sequence of reliability functions these times (M values of t_{RM}) can be either specified as input data or be timed to coincide with each inspection. In the latter case the distributions are calculated either just before or just after each inspection.

These three subsequences (any of which may be empty) are combined to form a total sequence of K values of time, t_k at which the reliability functions are evaluated.

3.6.1. Auxiliary functions

In addition to the basic reliability functions defined in Section 2.1, the following auxiliary functions are evaluated either as part of the time history of functions or together with the strength and failure densities at t_{RM} .

(1) Mean risk

For a given value of time the mean risk, $r_{mean}(t)$ is defined as the value of risk rate that would have produced the probability of survival at that time had that risk rate been applied as a constant risk over the interval $[0, t]$.

By equation (2.10)

$$P_S(t) = \exp\{-r_{mean}(t) \cdot t\} \quad (3.174)$$

so that,

$$r_{mean}(t) = -\log\{P_S(t)\}/t. \quad (3.175)$$

(11) Expected time of failure

Using \underline{F} to denote the random variable, time of failure, the expected time of failure $E(\underline{F})$ is given by,

$$E(\underline{F}) = \int_0^t t p_f(t) dt / P_F(t). \quad (3.176)$$

From the sequence of values of $p_f(t_k)$ and $P_F(t_k)$, $E(\underline{F})$ is computed by assuming that $p_f(t)$ varies exponentially over each time interval $[t_{k-1}, t]$, i.e.,

$$p_{\underline{F}}(t) = p_{\underline{F}}(t_{k-1}) \exp \left\{ \log \left\{ \frac{p(t_k)}{p(t_{k-1})} \right\} \frac{(t - t_{k-1})}{(t_k - t_{k-1})} \right\}, \quad t_{k-1} \leq t \leq t_k.$$

..... (3.177)

Integrating (3.177) over the interval $[t_{k-1}, t_k]$ leads to

$$E(\underline{F})_k = (tp_f)_k / p_{\underline{F}}(t_k) \quad (3.178)$$

$$\text{where} \quad (tp_f)_k = \int_0^{t_k} tp_{\underline{F}}(t) dt \quad (3.179)$$

and

$$\begin{aligned} (tp_f)_k &= (tp_f)_{k-1} + \frac{(t_k - t_{k-1})}{\log\{p_{\underline{F}}(t_k)/p_{\underline{F}}(t_{k-1})\}} \\ &\quad \times \left[t_k p_{\underline{F}}(t_k) - t_{k-1} p_{\underline{F}}(t_{k-1}) - \frac{(t_k - t_{k-1})[p_{\underline{F}}(t_k) - p_{\underline{F}}(t_{k-1})]}{\log\{p_{\underline{F}}(t_k)/p_{\underline{F}}(t_{k-1})\}} \right] \end{aligned}$$

..... (3.180)

The recurrence equation (3.180) can be used to construct the required sequence of estimates for $E(\underline{F})$. Note that each value (in particular the first) of $p_{\underline{F}}(t_k)$ must be non zero if equation (3.180) is to yield satisfactory results. Replacing a zero value by a small positive number (say 10^{-32}) will overcome this problem.

(iii) Distribution of strength

Given the density for strength $p_R(R | F > t)$, the distribution of strength is given by,

$$P_R(R | F > t) = \int_0^R p_R(R | F > t) dR \quad (3.181)$$

(iv) Distribution of the failing load

Given the failure density for strength, $p_R(R | t)$, the distribution of the failing load is given by

$$P_R(R | t) = \int_0^R p_R(R | t) dR. \quad (3.182)$$

(v) Expected value of strength

Given the density for strength $p_R(R | F > t)$, the expected value of strength $E(R | F > t)$ is,

$$E(R | F > t) = \frac{\int_0^{\infty} R p_R(R | F > t) dR}{p_R(\infty | F > t)} \quad (3.183)$$

(vi) Expected value of the failing load

Given the failure density for strength, $p_R(R | t)$, the expected value of the failing load is,

$$E(R | t) = \frac{\int_0^{\infty} R p_R(R | t) dR}{p_R(\infty | t)} \quad (3.184)$$

3.6.2. Inspection procedures

The inspection times can be either specified as input data or calculated by NERF.

(1) Periodic inspections

If the inspection times are specified, the inspection procedure is termed 'periodic'. Complete freedom is allowed regarding the selection of inspection times which do not have to coincide with the evaluation times. In fact, it is possible to perform a sequence of calculations for which only the inspection times are specified. Such a calculation is referred to loosely as a continuous inspection procedure wherein an inspection occur every time the reliability functions are evaluated.

(ii) Limit risk inspections

The calculation of inspection times by NERF uses an algorithm which monitors the total risk rate and arranges that inspections occur when the risk rate rises to a prescribed level. After each inspection the risk rate decreases, usually to a value lower than the prescribed value. In the event that this does not occur, the inspection procedure is terminated.

It is possible to prescribe a different limit risk for the first inspection from that used for the calculation of the remaining inspection times.

(iii) Periodic and limit risk inspections

A combination of the two inspection procedures is possible. The first inspection occurs at a specified time and the remaining inspections are computed by the limit risk algorithm.

(iv) Replacement at inspections

At each inspection a fraction of the population (magnitude $P_{det}(t_{i_j})$) is removed. This fraction can optionally be replaced by 'perfect' structures which will not experience any risk of failure. If this option is exercised, the probability of survival at any given time includes the sum of the probabilities of detection at all previous inspections, i.e.,

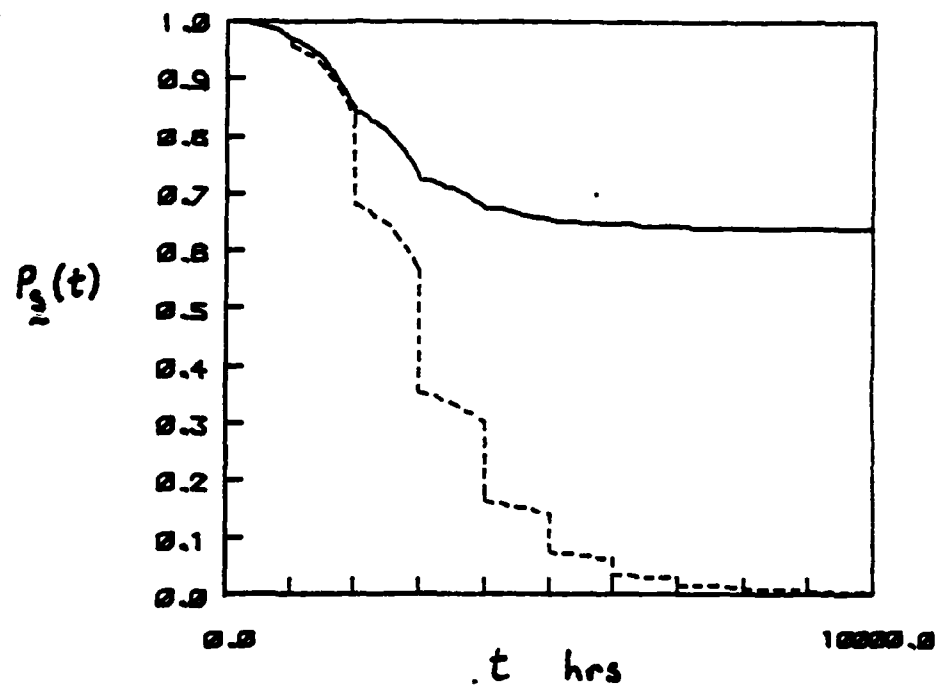
$$P_{\underline{S}}(t) = P_{\underline{S}}'(t) + \sum_{j=1}^{j'} P_{det}(t_{i_j}) \quad (3.185)$$

where t_{i_j}' is the time of the last inspection before time t and $P_{\underline{S}}'(t)$ the probability of survival without replacement.

Examples of the results of exercising the various inspection procedure options are shown in Figures 3.16 to 3.18. All calculations are for example A. Figures 3.16 and

3.17 show probabilities of survival and total risk rates respectively for two calculation sequences which both included periodic inspections but one exercised the replacement option whereas the other did not. Note that the calculation was continued to the point where, without replacement the whole population had either failed or been removed by inspection. The population receiving replacement structures finally asymptotes to a constant probability of survival indicating that the bulk of the population has been replaced by 'perfect' structures. (It should be emphasised here that in the course of normal operation of NERF such extreme calculations would rarely be made.)

Figure 3.18 shows two total risk rates obtained by the limit risk algorithm using crack length inspection tests having different values of a_d .



Replacement

No replacement

Figure 3.16. Probability of survival for example A subject to periodic inspections every 1000 hrs.

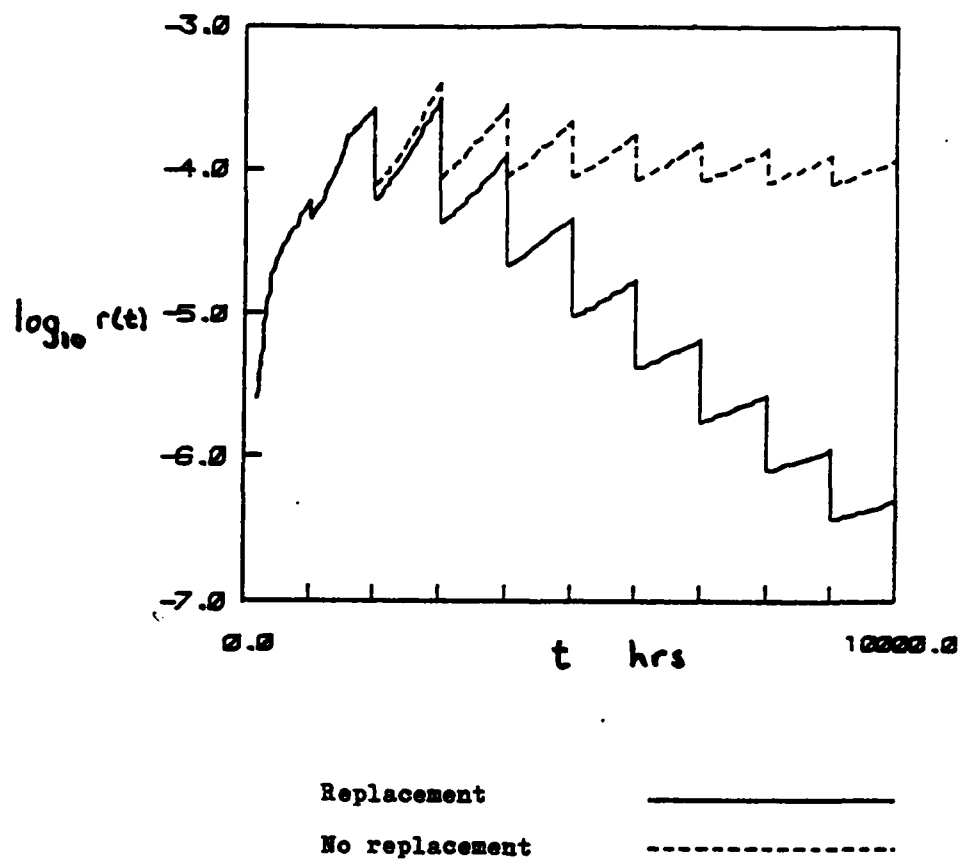


Figure 3.17. Total risk rate for example A subject to periodic inspections every 1000 hrs.

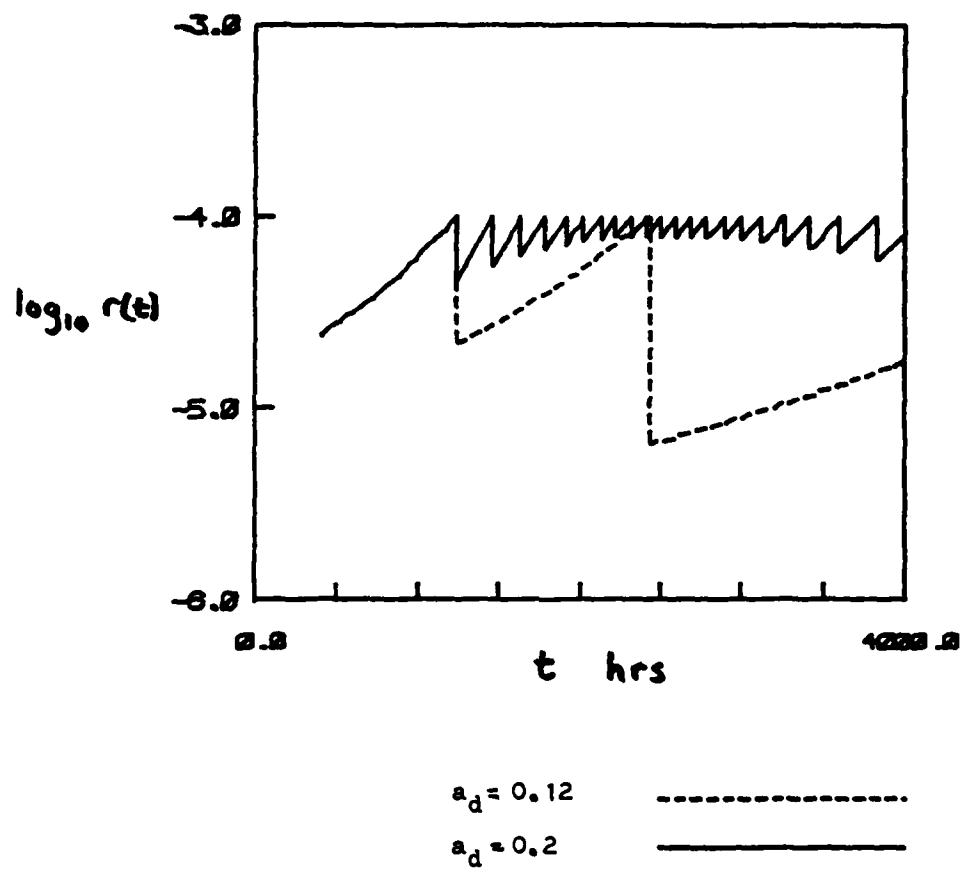


Figure 3.18 Total risk rate for example A for the case of limit risk inspection. Inspections are perfect and are based on two different crack length criteria.

3.6.3. Simplifying options

Although the numerical integration algorithms used by NERF are efficient, the calculation of the reliability functions can involve considerable computation, especially if the full generality offered by the computer program is employed. There is therefore sufficient motivation to seek computation reduction via simplifying approximations where appropriate. Two such approximations are described in this Section.

(1) Computation of $P_S(t)$ from $r(t)$

If the reliability functions are evaluated with sufficient frequency it may be possible to evaluate the probability of survival by integration of equation (2.6), i.e.,

$$P_S(t) = 1 - \int_0^t p_F(t) dt. \quad (3.186)$$

At each evaluation time an integral expression for $p_F(t)$ ($= r(t) \cdot P_S(t)$) is available. Equation (3.186) can be approximated by assuming the $p_F(t)$ varies exponentially over each time interval, i.e., (ref equation (3.177))

$$p_F(t) = p_F(t_{k-1}) \exp \left\{ \log \{ p_F(t_k) / p_F(t_{k-1}) \} (t - t_{k-1}) / (t_k - t_{k-1}) \right\} \quad (3.187)$$

.....

which leads to the recurrence relation,

$$P_S(t_k) = P_S(t_{k-1}) - \frac{(t_k - t_{k-1})(p_F(t_k) - p_F(t_{k-1}))}{\log \{ p_F(t_k) / p_F(t_{k-1}) \}}. \quad (3.188)$$

Equation (3.188) can be used to construct an approximate sequence of values for $P_{\underline{S}}(t)$. Provided the time intervals are small, these approximate values may be sufficient particularly in those cases where interest is confined to small values of time when $P_{\underline{S}}(t) \approx 1$.

One increase in computational effort resulting from the use of (3.188) is that associated with the evaluation of the probability of detection at each inspection. Whereas the use of integral expressions for $P_{\underline{S}}(t)$, (or $P_{\underline{F}}(t)$) permits the evaluation of $P_{\text{det}}(t_{i,j})$ by taking the difference between the probabilities of survival before and after an inspection, the approximate sequence cannot yield this information and an integral expression for $P_{\text{det}}(t_{i,j})$ must be used. Equation (3.188) loses its computational advantage if inspection times dominate the total sequence of evaluation times.

(11) Neglecting the effect of losses from the population

The calculation of the loss factor, $H(\alpha, \beta, n_0, t)$ involves an integration, which may be avoided if the factor is omitted. Various approximate expressions which neglect the effect of losses from the population were presented by Diamond and Payne³ and NERF permits $H(\alpha, \beta, n_0, t)$ to be neglected for all the reliability functions. Acceptable results can be obtained when $P_{\underline{S}}(t) \approx 1$. However, the use of an interpolation table for the loss factor, largely obviates the need to invoke this facility which is retained primarily to ensure compatibility with earlier calculations.

4. NUMERICAL METHODS

The organisation, operation and performance of NERF depends heavily on the numerical methods used to evaluate the reliability functions. In particular, the selection of the method for numerical integration has a fundamental effect on the structure of the code, the efficiency of the calculations and, in terms of the functional forms of the density and input functions, the versatility of the computer program.

An adaptive integration technique which automatically adjusts the points of evaluation of the integrand to control numerical errors was chosen and has proven to be efficient, reliable and to a large extent has removed any dependence of the code on the forms of the density and input functions.

The adaptive integration algorithms form the heart of NERF which can be considered as a computer program which organises the reliability models and data so that they can be processed by the integration algorithms, (in much the same way that a finite element program organises the element models for processing by algorithms for the solution of sets of equations.)

The requirement that the input functions be represented by ordered pairs of argument and function values necessitated the use of an efficient and well behaved (especially in terms of continuity) interpolation methods. Cubic spline interpolation has proved to be satisfactory.

The calculation of inspection intervals using a limit risk condition (Section 3.6.2) demanded the use of an equation solver which could cope with the functions defined by the expressions for total risk rate, $r(t)$. The secant method is simple and effective in this application and is used elsewhere in the code to provide function inverses and solve equations governing some integration limits.

Other numerical methods used by NERF include the location of sensible limits for the arguments of probability density functions and processes involving sequences of numbers. (such as index location and merging of two sequences.)

All these numerical methods are general in nature and are not particularly tied to the evaluation of reliability functions. For this reason they are described separately from the main documentation of the NERF program (Chapter 5). By accessing the NERF program as a library of subroutines and functions, these numerical methods are available to other programs.

4.1. Numerical Integration

As already stated, the selected method of numerical integration dictates much of the structural characteristics of the NERF program. When the development of the numerical methods for the evaluation of reliability functions was commenced at ARL, adaptive integration methods were relatively new, particularly those that could be applied to multiple integrations. A pilot study using a conventional, non-adaptive, composite Simpson method demonstrated that the peaked nature of many of the integrands of the reliability functions necessitated the use of an adaptive method.

Adaptive methods for integration can automatically select the points of evaluation of the integrand and the integrating functions. According to Rice¹⁴, there are somewhere between 10^6 and 10^7 algorithms that are potentially interesting and significantly different from one another. In recent years there have appeared interesting studies of various adaptive algorithms and methods for selecting the most appropriate algorithm for a given application.

With the exception of a recent discussion by Fritsch, Kahaner and Lyness,¹⁵ rigorous discussions of adaptive algorithms and error estimation have been confined to single levels of integration. The problem of interfacing adaptive integration algorithms designed for the evaluation of single integrals to form a suitable algorithms for multiple integration has been largely open, certainly

during the period of time during which NERF was developed.

The design of an algorithm for application to the evaluation of reliability functions consists of two steps. The first is the construction of an algorithm for single integration; the second is the development of suitable interfacing for the application of that algorithm to a multiple integration consisting of nested single integrations. In particular, the accumulation of error estimates and the control of inner integrations to satisfy the requirements of the outer integrations require careful consideration. The description of the adaptive algorithms used by NERF follows these two steps. The mathematical basis and implementation of the algorithm for single integration is described first followed by the description of the interfacing for multiple integration.

4.1.1. Mathematical basis (single integration)

The algorithm used for single integration is based on the adaptive Simpson algorithm originally developed by McKeeman¹⁶. Subsequent refinements and error analyses were presented by Lyness¹⁷. The discussion presented here is a blend of that of Lyness and more recent discussions of error estimates by Osborne¹⁸ and extrapolation techniques by Robinson¹⁹.

The objective of any numerical integration procedure is to estimate,

$$I(a,b) = \int_a^b f(x) dx \quad (4.1)$$

within a specified absolute tolerance, ϵ_{abs} . Most procedures satisfy this objective. However, the important consideration in any practical situation is computing cost and the ideal integration procedure is one which satisfies the above objective with minimum cost, (usually expressed in terms of the number of integrand evaluations.) Cost minimisation is particularly important if the procedure is to be nested to provide a multiple integration algorithm.

It is not possible to establish rigorously the optimal qualities of a given adaptive algorithm. Moreover, it is not possible to prove that a given algorithm will cope with all eventualities (see Lyness and Kaganove²⁰). In fact, it is preferable, from the cost viewpoint, to operate most adaptive algorithms under conditions for which there is a finite probability of failure of the integration algorithm. (Failure manifests as

an infinite cost to satisfy (4.1).

Fortunately, this probability of failure can be kept very small and the error estimates provided during the operation of the algorithm usually give a ready indication of incipient failure. Moreover, the rising cost associated with failure gives the adaptive procedure the desirable property of providing a self indication of possible failure.

Let $I_n(a,b)$ denote a numerical estimate of $I(a,b)$ using n evaluations of the integrand. The cost objective can be expressed as finding the minimum value of n for which,

$$|I(a,b) - I_n(a,b)| \leq \epsilon_{\text{abs}}. \quad (4.2)$$

An adaptive integration procedure satisfies (4.2) by either adjusting the points in a,b where the integrand, $f(x)$, is evaluated or by selecting different rules. Note that the evaluation count, n , includes all function evaluations made during the selection process and not just those used to form the final answer. The algorithms described here adjust only the points of integrand evaluation.

Condition (4.2) is an absolute error criterion. An alternative relative error criterion can be used, i.e.,

$$|I(a,b) - I_n(a,b)| \leq \epsilon_{\text{rel}} \left| \int_a^b f(x) dx \right|. \quad (4.3)$$

The application of (4.3) within the adaptive algorithm poses the problem of providing estimates of the integral term which is, of course, not known precisely until the integration process is completed.

In practice, (4.3) is replaced by,

$$|I(a,b) - I_n(a,b)| \leq \epsilon_{\text{rel}} \int_a^b |f(x)| dx. \quad (4.4)$$

A running estimate of the integral term is kept during the operation of the algorithm. Note that although (4.4) is used approximately during the algorithm, error estimates can be made on an absolute basis and converted to relative error estimates on completion of the integration.

The distribution of integrand evaluation points is found by successive subdivision of the interval $[a,b]$. As each subdivision is made, the change in the estimate for the integral is examined in the light of either (4.2) or (4.4) and a decision made regarding further subdivision.

McKeeman's¹⁶ original scheme was based on the 3 point Simpson rule. Denoting the estimate obtained by applying this rule over the whole interval by $S_3^1(a,b-a)$, where,

$$S_3^1(a,b-a) = \frac{b-a}{6} \{f(a) + 4f((a+b)/2) + f(b)\} \quad (4.5)$$

where the notation is defined by,

$$S_3^F(x, \Delta x_r) = \frac{\Delta x_r}{6} \{f(x) + 4f(x + \Delta x_r/2) + f(x + \Delta x_r)\} \quad (4.6)$$

where Δx_r depends on the level subdivision, with Δx_1 being the same as $(b-a)$. McKeeman's subdivision logic was based on dividing each interval into 3 sub-intervals. For the first interval, $[a,b]$,

the estimate after subdivision is given by,

$$\begin{aligned}
 S_7^1(a, b-a) &= S_7^1(a, \Delta x_1) \\
 &= \frac{\Delta x_1}{18} \left\{ f(a) + 4f(a+\Delta x_1/6) + 2f(a+\Delta x_1/3) + 4f(a+\Delta x_1/2) \right. \\
 &\quad \left. + 2f(a+2\Delta x_1/3) + 4f(a+5\Delta x_1/6) + f(a+\Delta x_1) \right. \\
 &\quad \left. \dots \right. \quad (4.7)
 \end{aligned}$$

$$= S_3^2(a, \Delta x_2) + S_3^2(a+\Delta x_2, \Delta x_2) + S_3^2(a+2\Delta x_2, \Delta x_2) \quad (4.8)$$

where

$$\Delta x_r = (b-a)/3^{r-1}. \quad (4.9)$$

Considering the first subdivision, if $|S_3^1(a, \Delta x_1) - S_7^1(a, \Delta x_1)|$ is greater than some chosen criterion, then the subdivision process continues by dividing each of the intervals $[a, a+\Delta x_2]$, $[a+\Delta x_2, a+2\Delta x_2]$, $[a+2\Delta x_2, a+3\Delta x_2]$ in the same manner as described above for the first subdivision. The criterion applied at each level of subdivision depends on Δx_r . Mckeeman used the criterion $\epsilon/3^{r-1}$ where ϵ is the absolute error requirement (4.2). In his scheme subdivision continued if,

$$|S_7^r(x, \Delta x_r) - S_3^r(x, \Delta x_r)| > \epsilon/3^{r-1} \quad (4.10)$$

Lyness¹⁷, suggested several modifications to this logic: those that have been implemented in the NERF algorithm are described below.

- (1) Each interval is bi-sected rather than tri-sected.

The estimate after subdivision of the interval $[x, x+\Delta x_r]$ is

$$\begin{aligned}
S_5^r(x, \Delta x_r) &= \frac{\Delta x_r}{12} \left\{ f(x) + 4f(x + \Delta x_r/4) + 2f(x + \Delta x_r/2) \right. \\
&\quad \left. + 4f(x + 3\Delta x_r/4) + f(x + \Delta x_r) \right\} \\
&= S_3^{r+1}(x, \Delta x_{r+1}) + S_3^{r+1}(x + \Delta x_{r+1}, \Delta x_{r+1})
\end{aligned}
\tag{4.11}$$

where

$$\Delta x_r = (b-a)/2^{r-1} \tag{4.12}$$

(ii) Equation (4.10) is replaced by

$$|S_5^r(x, \Delta x_r) - S_5^r(x, \Delta x_r)| > C_0 \epsilon / 2^{r-1} \tag{4.13}$$

where C_0 is a constant, defined below.

(iii) After the subdivision process is complete, extrapolation is used to provide a more accurate estimate of the integral and to estimate the error.

The question of extrapolation was first considered by Lyness¹⁷ and more complete analyses were presented by Osborne¹⁸ who derived a generalised Euler Maclaurin expansion for the error associated with a numerical integration formula. As simplified by Robinson¹⁹, the results and implications for the present scheme are described below.

Let Q_n be an n point quadrature sum defined by,

$$Q_n \{f(x); a, b\} = h \sum_{j=1}^n A_j f(a + \xi_j h) \tag{4.14}$$

where $h = (b-a)$ and $0 \leq \xi_1 < \xi_2 < \dots < \xi_n \leq 1$ and the A_j are positive weights that are independent of f and h . If the function f

is continuous and has continuous derivative of sufficiently high order, then it can be shown (Osborne¹⁸) that

$$I(a,b) = Q_n\{f(x), a, b\} - \gamma_{k+1} h^k [r^{k-1}(b) - r^{k-1}(a)] + o(h^{k+1}) \quad \dots (4.15)$$

The coefficient γ_{k+1} is the first non zero term in the series defined by

$$\gamma_1 = \frac{1}{(1-1)!} \sum_{j=1}^n \Lambda_j (\xi_j)^{1-1} - \sum_{j=2}^i \gamma_{j-1} \xi_j / j! \quad (4.16)$$

with $\gamma_1 = 1$.

For the 3 point Simpson rule, $n=3$, $\xi_1=0$, $\xi_2=\frac{1}{2}$, $\xi_3=1$, $\Lambda_1=1/6$, $\Lambda_2=4/6$, $\Lambda_3=1/6$ and the first nonzero γ_1 is γ_5 given by

$$\begin{aligned} \gamma_5 &= \frac{1}{4!} \left[\frac{4}{6} \left(\frac{1}{2}\right)^4 + \frac{1}{6} - \frac{1}{5} \right] \\ &= \frac{1}{2880} \end{aligned} \quad (4.17)$$

so that

$$I(a,b) = S_3^1(a, \Delta x_1) - \frac{h^4}{2880} [r^3(b) - r^3(a)] + o(h^5) \quad (4.18)$$

Similarly,

$$I(a,b) = S_5^1(a, \Delta x_1) - \frac{h^4}{46080} [r^3(b) - r^3(a)] + o(h^5) \quad (4.19)$$

and

$$I(a,b) = S_5^1(a, 4\Delta x_1) - [S_3^1(a, \Delta x_1) - S_5^1(a, \Delta x_1)]/15 + o(h^5) \quad (4.20)$$

This result is true for integration over any interval, i.e.,

$$I(x, x+\Delta x_F) = S_5^F(x, \Delta x_F) - [S_3^F(x, \Delta x_F) - S_5^F(x, 4\Delta x_F)]/15 + o(\Delta x_F^5) \quad (4.21)$$

If the interval $[x, x+\Delta x_r]$ is accepted by the subdivision logic,

$S_5^r(x, \Delta x_r)$ will be the term representing the integral over that interval. The final result will be the sum of such integrals.

The term $S_3^r(x, x_r) - S_5^r(x, x_r)$ has already been computed for application in equation (4.13) and can be used to return an extrapolated estimate via equation (4.21). This estimate will have an error of order Δx_r^5 . Lyness argued that the original acceptance criterion is conservative if extrapolation is used and recommended that (4.13) be applied with $\bar{C}_0 = 15$. The second term on the R.H.S. of equation (4.21) is taken as the error estimate for that interval. The error estimate for the whole integral is then

$$\epsilon_{\text{est}} = \left| \sum_{\text{accepted intervals}} (S_3^r(x, \Delta x_r) - S_5^r(x, \Delta x_r)) \right| / 15 \quad (4.22)$$

Note that if the sum in equation (4.22) is made as the intervals are accepted the extrapolation can be made, once only, to the final result rather than when each interval is accepted.

4.1.2. Implementation (single integration)

The adaptive integration algorithm coded in NERF is based on the adaptive 3 point Simpson algorithm described above. Several modifications to the algorithm have, however, been implemented.

(1) Specification of nodes for initial subdivision

Some of the integrand functions are defined by sequences of ordered pairs of argument and function values. The interpolation algorithms do not necessarily result in continuous 3rd and 4th derivatives, meaning that the error analysis and extrapolation arguments presented in the previous section do not strictly apply. During the development of the NERF program, it was observed that one of the reasons for failure of the integration algorithm was interference between the intervals set by the ordered pairs defining the integrand and those created by the subdivision logic used by the integration algorithm.

To circumvent this difficulty, an initial set of intervals can be specified before entry into the adaptive integration algorithm; i.e., the interval $[a, b]$ is divided by x_i , $i = 1, \dots, m$, where $x_1 = a$, $x_m = b$ and $a \leq x_i < x_{i+1} \leq b$. Each interval $[x_i, x_{i+1}]$ is integrated separately so that the specified nodes appear as individual end points for the adaptive integration. Over each interval, the integrand is assumed to have continuous derivatives up to 4th order. The overall integral is given by,

$$I(a, b) = \sum_{i=1}^{m-1} I_1(x_i, x_{i+1}) \quad (4.23)$$

If each integration has an error ϵ_1 , say then the total error ϵ is given by,

$$\begin{aligned}\epsilon &= \left| \sum_{i=1}^{n-1} \epsilon_i \right| \\ &\leq \sum_{i=1}^{n-1} |\epsilon_i| \quad (4.24)\end{aligned}$$

The requirement $\epsilon \leq \epsilon_{\text{abs}}$ can be met if

$$\epsilon_1 \leq \epsilon_{\text{abs}}(x_{1+1} - x_1)/(b-a). \quad (4.25)$$

(11) Integration over initial subdivision intervals

Each interval $[x_1, x_{1+1}]$ is integrated initially using the trapezoidal rule, viz ,

$$T_2^1(x_1, x_{1+1} - x_1) = \frac{(x_{1+1} - x_1)}{2} [f(x_1) + f(x_{1+1})]. \quad (4.26)$$

The interval is then subdivided to produce the estimate,

$$\begin{aligned}T_3^1(x_1, x_{1+1} - x_1) &= T_2^2(x_1, \frac{x_{1+1} - x_1}{2}) + T_2^2(\frac{x_1 + x_{1+1}}{2}, \frac{x_{1+1} - x_1}{2}) \\ &\dots \dots \dots (4.27)\end{aligned}$$

Using equations (4.15) to 4.16),

$$\begin{aligned}I_1(x_1, x_{1+1}) &= T_2^1(x_1, x_{1+1} - x_1) - \frac{h^2}{12} [f'(b) - f'(a)] + o(h^3) \\ &\dots \dots \dots (4.28)\end{aligned}$$

$$\begin{aligned}&= T_3^1(x_1, x_{1+1} - x) - \frac{h^2}{48} [f'(b) - f'(a)] + o(h^3). \\ &\dots \dots \dots (4.29)\end{aligned}$$

Using these two estimates, the extrapolation for the interval is given by,

$$I_1(x_1, x_{1+1}) = T_3^1(x_1, x_{1+1} - x_1) - \frac{1}{3} [T_2^1(x_1, x_{1+1} - x_1) - T_3^1(x_1, x_{1+1} - x_1)] + o(h^4) \quad (4.30)$$

$$= S_3^1(x_1, x_{1+1} - x_1) + o(h^4). \quad (4.31)$$

The extrapolation from the two trapezoidal estimates is the same as a 3 point Simpson integration over the interval. Equation (4.31) would, of course, be applied directly. However, the sequence adopted above yields, via the term $T_3^1(x_1, x_{1+1} - x_1) - T_2^1(x_1, x_{1+1} - x_1)$, a measure of the error associated with the Simpson integration and a means for determining whether the interval requires the full adaptive integration. If,

$$\frac{1}{3} |T_3^1(x_1, x_{1+1} - x_1) - T_2^1(x_1, x_{1+1} - x_1)| \leq \epsilon_{\text{abs}} (x_{1+1} - x_1) / (b-a) \quad (4.32)$$

the initial integration is accepted for the interval $[x_1, x_{1+1}]$. If the integration is accepted, the L.H.S. of equation (4.32) is taken as the estimate of the error, ϵ_1 .

An interval for which (4.32) is not satisfied is integrated by the adaptive 3 point Simpson algorithm. In this case, the error, ϵ_1 , is calculated via equation (4.22).

The overall subdivision sequence is shown schematically in Figure 4.1. Note that the initial subdivision nodes are obtained from a specified set which encompasses the interval $[a, b]$. The code in the adaptive integration routines issues a warning if this is not so.

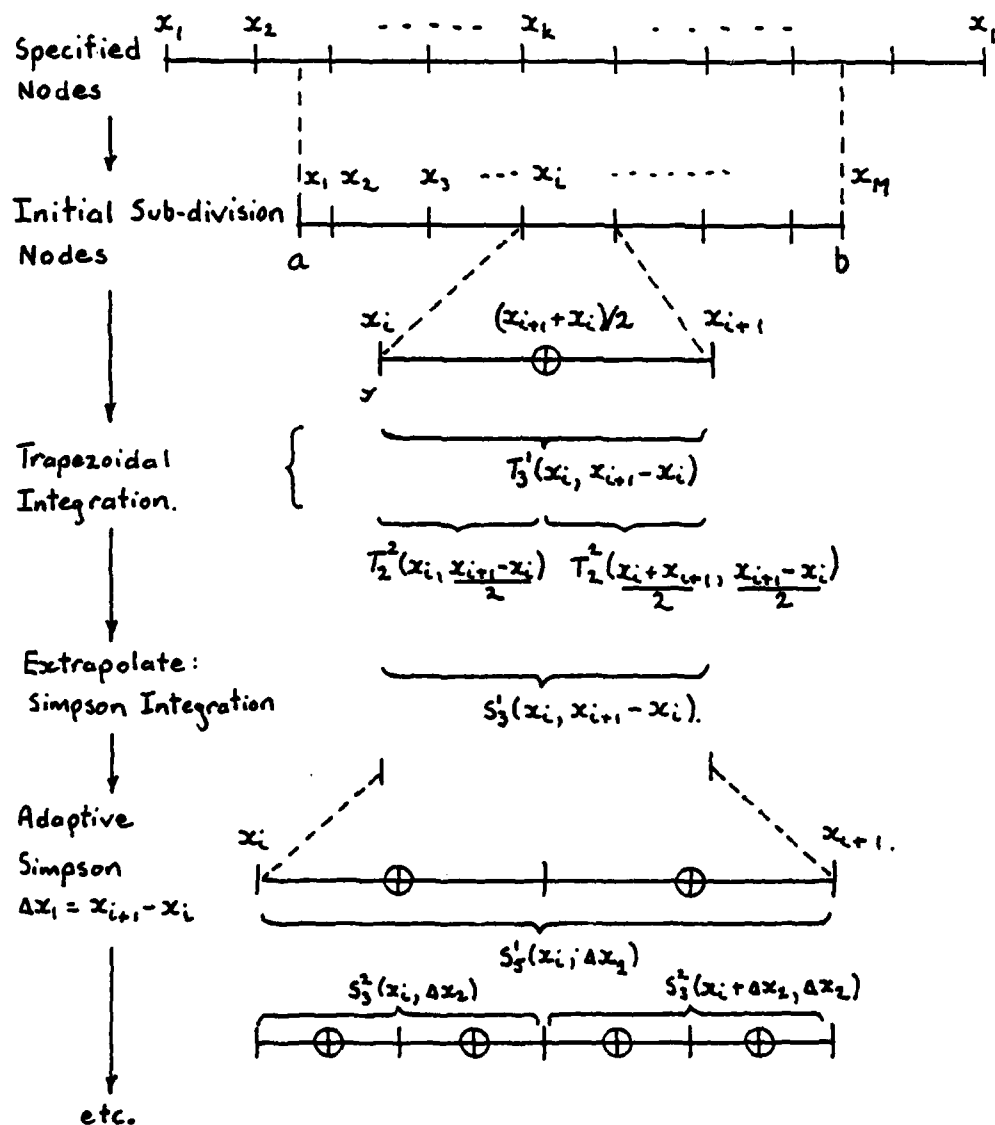


Figure 4.1. Schematic representation of the overall adaptive integration process. The points denoted by vertical bars are locations of integrand values that are available from previous steps. Circled points indicate new integrand evaluations. The subdivision logic within the Simpson algorithm is detailed more fully in Figure 4.2.

The subdivision sequence within the adaptive Simpson algorithm is shown in more detail in Figure 4.2. The broken lines enclose the terms which are evaluated for a given subdivision. New function evaluations are indicated by the circles while the function evaluations that have been made at a previous level of subdivision are indicated by the vertical lines. The symbols for the integration approximations are those defined by equations (4.6) and (4.11) with the dependence on x and Δx_r omitted. Note that the adaptive integration is now performed over the interval $[x_1, x_{1+1}]$ so that

$$\Delta x_r = (x_{1+1} - x_1) / 2^{r-1} . \quad (4.33)$$

(11) Relative Error Criterion

In practice, the absolute error criterion, ϵ_{abs} is replaced by the relative error criterion defined by equation (4.3). The logic described in the previous sections then applies if ϵ_{abs} is replaced by the R.H.S. of equation (4.4), (Lyness¹⁷). The adaptive integration algorithm coded here can accept two error criteria, ϵ_{abs} and ϵ_{rel} . The criterion (ϵ or ϵ_{abs}) in equations (4.13), (4.25) and (4.32) are replaced by ϵ' where

$$\epsilon' = \max \left\{ \epsilon_{abs}, \epsilon_{rel} \int_a^b |f(x)| dx \right\} . \quad (4.34)$$

meaning that the weaker of the two conditions is taken.

The absolute criterion is used only when a given algorithm is being used to evaluate an inner integration of a multiple integration and can not be set explicitly.

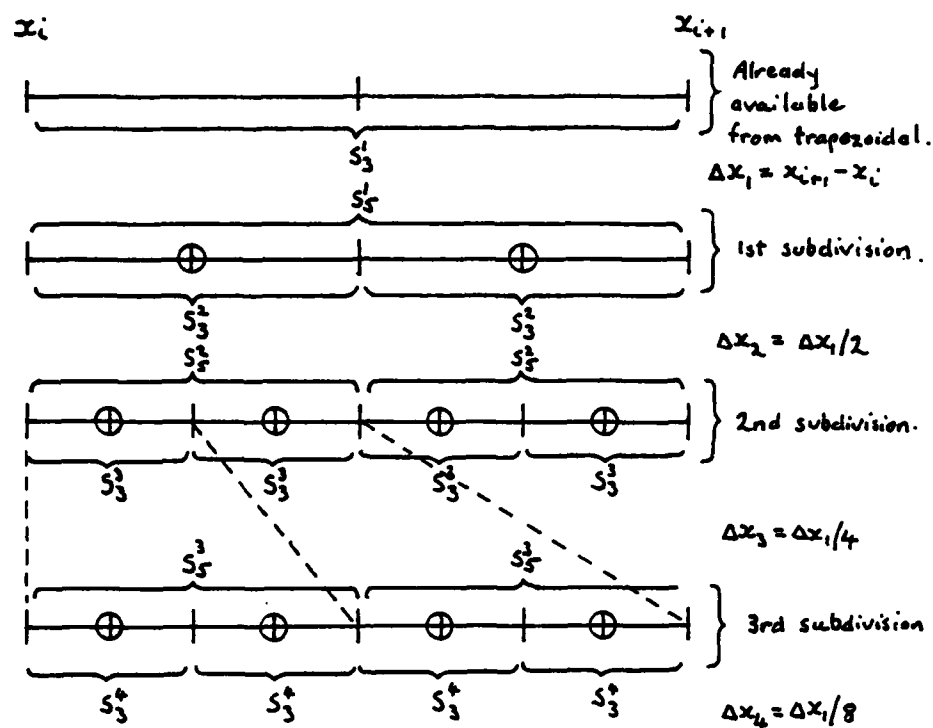


Figure 4.2. Schematic representation of the sequence of subdivisions made by the adaptive Simpson integration algorithm. At each level, the circled points indicate new integrand evaluations for that level.

As already mentioned, the integral term in (4.34) is known only approximately during the adaptive integration. The estimate is updated continuously. An initial estimate is made after the integrations over the initial subdivisions have been completed, i.e.,

$$\int_a^b |f(x)| dx \simeq \sum_{i=1}^{m-1} |s_3^1(x_1, x_1, -x_1)| \quad (4.35)$$

The code ensures that $m \geq 5$ so that at least 9 integrand evaluations are used to make the initial estimate. If insufficient initial subdivision nodes are specified then 5 equally spaced nodes are automatically generated.

If the estimate in equation (4.35) is numerically insignificant, ($< 10^{-32}$) the integral is assumed to be zero and no further calculations are made.

(iv) Adaptive subdivision logic

Examination of Figure 4.2 reveals that failure to satisfy condition (4.23) for any interval necessitates subdivision to generate 2 new intervals involving 4 new integrand evaluations. Taking the subdivision resulting from the failure of s_3^1 and s_5^1 to satisfy (4.23) for the interval Δx_1 , two new intervals of length Δx_2 are generated.

In the process of generating S_5^2 for each interval, the estimates S_3^3 are also generated. Either of the two Δx_2 intervals may fail to satisfy (4.23): the particular interval is subdivided again. The process is continued until, generally, (4.23) is satisfied. If, at any level, both intervals fail to satisfy (4.23), then the interval for which the difference between S_3^r and S_5^r is the greater is subdivided first. The other interval remains as a candidate for subdivision at a later stage.

Subdivision cannot continue indefinitely. In the NERF algorithms subdivision ceases when r reaches a specified maximum level (selected by the user). Space allocation within the integration code limits this maximum to 30, so that the smallest interval will be $(x_{i+1} - x_i)/2^{29} = (x_{i+1} - x_i) \times 1.86 \times 10^{-9}$. In practice, the precision of the computer imposes a limit to the possible level of subdivision. Lyness¹⁷ detected problems arising from numerical roundoff by examining the sequence of estimates, S_3^r for a given interval. Departure from a monotonic approach to a limit was taken as an indication of roundoff errors. In the present algorithm, subdivision ceases when

$$\Delta x_r < (b - a) \times 10^{-7} \quad (4.36)$$

The decisions made during the adaptive subdivision follow the logic shown in Figure 4.3.

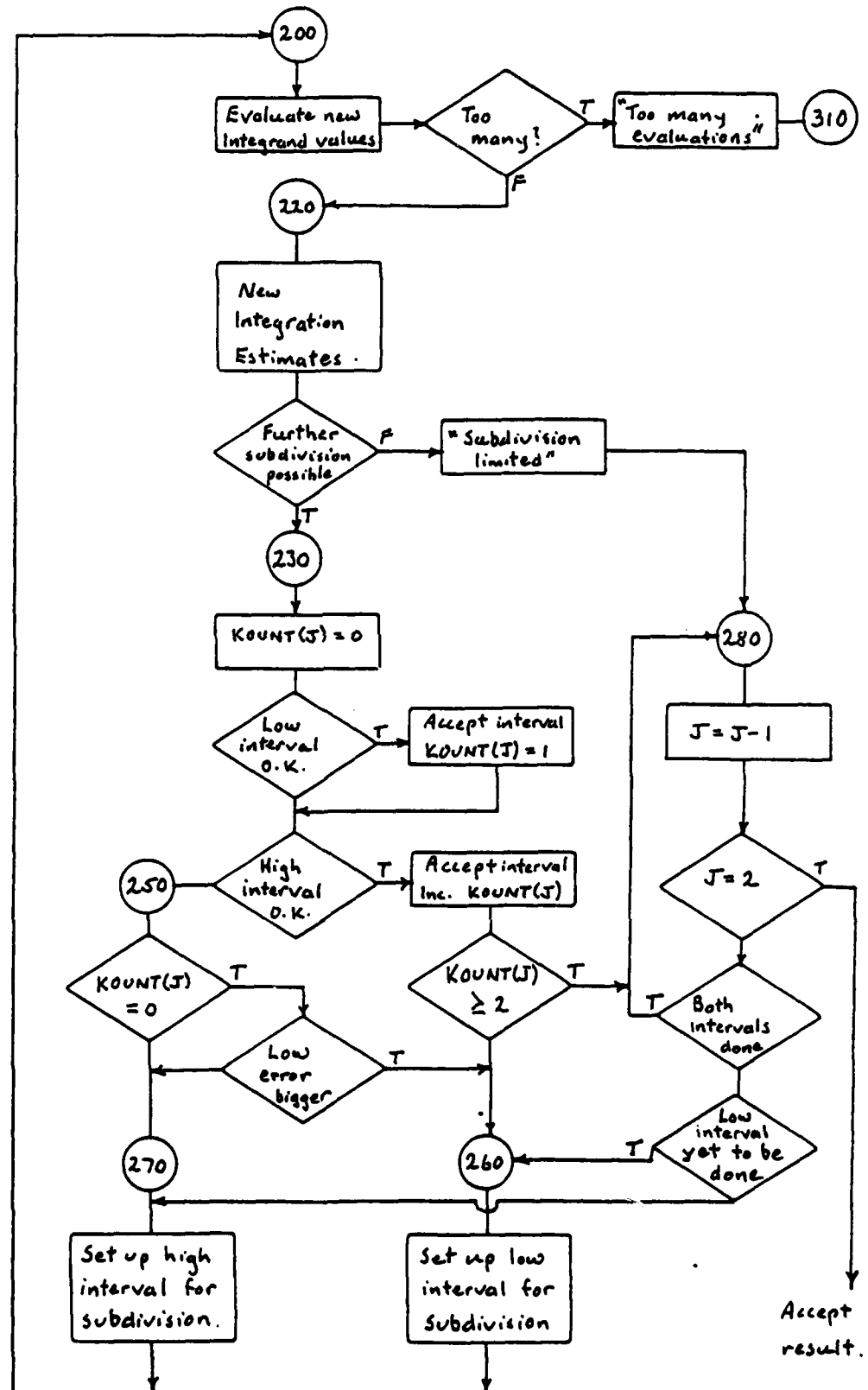
Start with $J=3$ 

Figure 4.3. Logic associated with the subdivision process in the adaptive Simpson algorithm.

The evaluation of a single integral is controlled by the function sub-program ADAPT2. This function is the lowest in a hierarchy of three functions, ADAPTO, ADAPT1 and ADAPT2 designed to evaluate multiple integrals. In principle, each could be used for a single integration, but ADAPT2 involves the least additional code and, therefore, superfluous computation.

These 'main routines' are supported by the following subroutines.

ADASET sets up constants used by the three functions and initialises parameters which depend on the desired error criterion. (Note that the same error criterion is used by all three integrating functions.)

ERROUT writes error or warning messages as triggered by any of the three main routines.

INFST (and entry points INFLE1, INFLE2 and INFSUP) together with INFINT write data regarding the overall evaluation of an integral.

Only ADASET, ERROUT and ADAPT2 will be described in this section. The remaining functions and subroutines will be described following the discussion of the techniques used for multiple integration.

Each of the integrating functions assumes that a suitable integrand function is available. The name of this function is passed as a parameter to the integrating function.

For example, suppose the adaptive integration function ADAPT2 is to be used to evaluate

$$I(1,5) = \int_1^5 x^2 dx \quad . \quad (4.37)$$

A function sub-program SQRX(X) supplies the square of X. The essential code to evaluate the integral is;

EXTERNAL SQRX; Identify SQRX as a function to be passed as a parameter.

CALL ADASET(.001); Set the relative error criterion to .001.

ANS = ADAPT2(SQRX,1.0,5.0,DUM,0,'SQRX','X',1.0,ERROR)

The parameters in the call are identified below. The value of the integral is returned as the value of the function and the error estimate (absolute). Essential information regarding the progress of the integral and final counts of function evaluations will be sent, via the subroutine PROMPT, to the control terminal and to logical unit number 4.

FUNCTION ADAPT2(F,XMIN1,XMAX1,XP1,NP1,FINTG,VARNAM,FCTR,ERROR)

Function

ADAPT2 returns, as the value of the function, the integral

$$I(a,b) = \int_a^b f(x)dx.$$

The function ADAPT2 is configured to operate as either a single integration algorithm or the innermost integration of a multiple integration algorithm.

Parameter List

F: Function defining the integrand, $f(x)$.

XMIN1: Lower limit of integration, a .

XMAX1: Upper limit of integration, b .

XP1: Array containing possible boundaries for initial subdivisions.

NP1: Number of values in XP1. If NP1 is greater than 2 the values in XP1 must encompass XMIN1 and XMAX1. If NP1 is less than 3, ADAPT2 sets up five equally spaced nodes for the initial subdivision.

FINTG: Five character string identifying the integrand

VARNAM: Five character string identifying the integration variable.

FCTR: Factor by which the integration is to be multiplied by to form the integrand for the next level of integration. For a single integration FACTR=1.0.

ERROR: Returned value of the error estimate multiplied by FCTR.

Control Parameters Stored In Common

The following variables exercise control over the adaptive integration algorithm.

NVMAX : (in INFOR2) Sets the maximum number of integrand evaluations to be used in any single integration step.

MAXLEV: (in INFOR2) Sets the maximum level of subdivision (i.e. max value of r) used by the adaptive Simpson algorithm.

Operation

The function ADAPT2 consists of two main sections. The first sets up the initial subdivisions according to the nodes stored in XP1. The initial integrations are then made, as described in Section 4.1.2 part (ii), and outlined schematically in Figure 4.1.

The second section performs the adaptive Simpson integration according to the logic described in Sections 4.1.1 and 4.1.2 part (iv).

This function is largely free of code associated with the interfacing for multiple integration. The essential section of such code is at the beginning of the function. If either outer integration function (ADAPTO or ADAPT1) is currently active, an absolute error criterion is calculated (see Section 4.1.4).

The three adaptive integration functions are the largest

sub-programs in the NERF computer program. It is not practical to detail the operation of the FORTRAN code more than has already been done in Sections 4.1.1 and 4.1.2. A complete list of variables appearing in the code is given Table 4.1. This should assist the programmer to relate the code to the mathematical description. Note that major statement numbers have been included in Figure 4.3.

Figure 4.4. shows the major program variables associated with obtaining the Simpson rule integration estimates for the subdivisions. The storage of integrand evaluations is described schematically in Figure 4.5.

Note that Table 4.1. does not list variables in ADAPTO and ADAPT1 that are associated with the accumulation of error terms for multiple integrations. Some variables and common blocks end with a number (0, 1 or 2). In the table these names end with 2; similar descriptions apply for variables or common blocks ending with the other numbers.

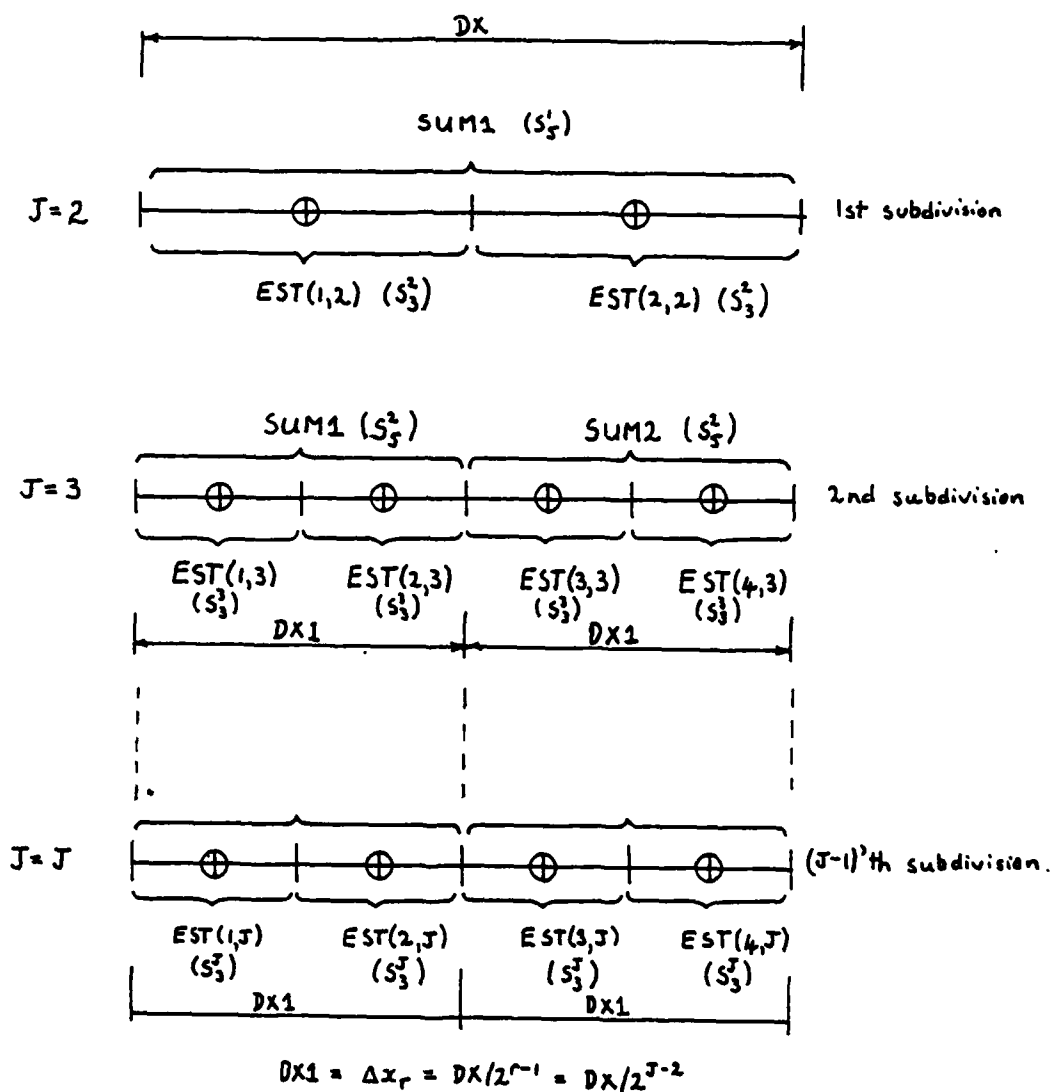


Figure 4.4. Program variables and arrays associated with the integration estimates. This figure should be studied in conjunction with Figure 4.2.

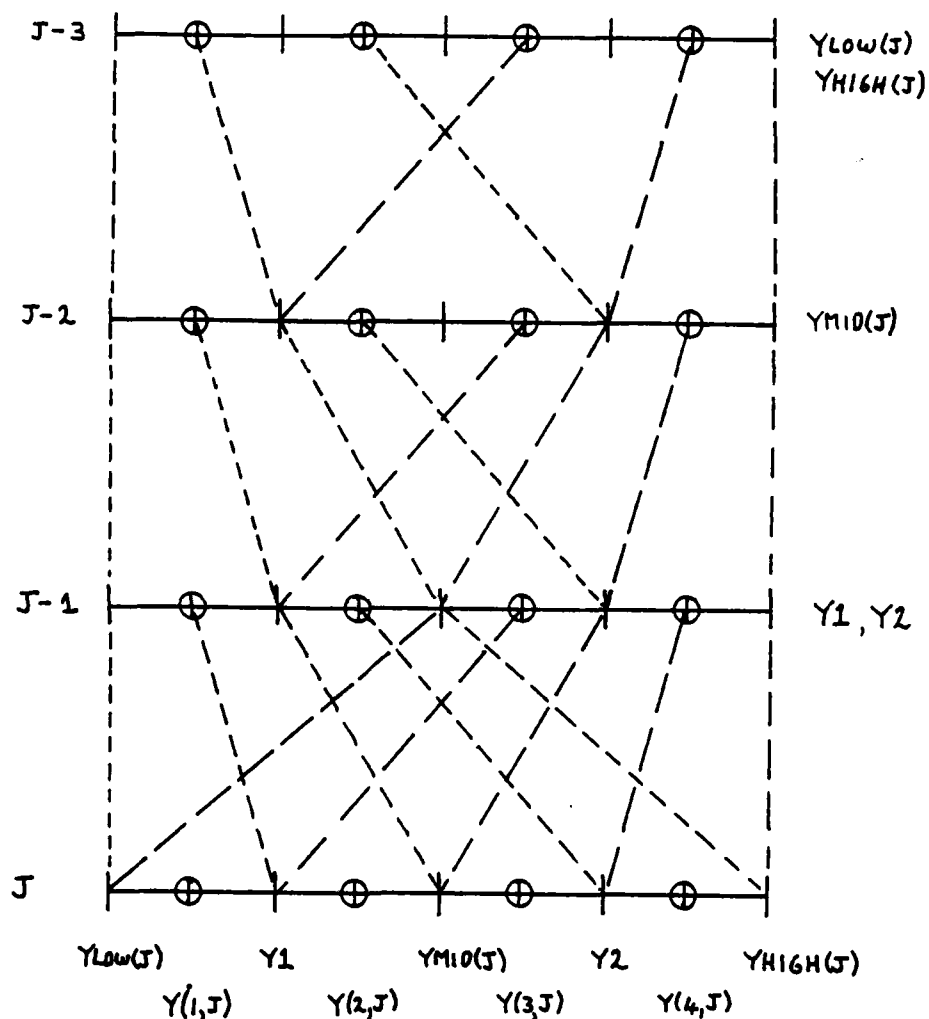


Figure 4.5. Program variables and array names associated with the storage of integrand values used to make estimates of the integrations over a J 'th level subdivision. New values are stored in the $Y(I,J)$ array. Existing values are stored in the $YLOW$, $YMID$ and $YHIGH$ arrays and the variables $Y1$ and $Y2$. These values are linked to $Y(I,J)$ values at previous levels as shown. (The level for each existing value is indicated on the right hand side of the diagram.)

Table 4.1.

Variables Used in the Adaptive
Integration Functions - ADAPTO,
ADAPT1 and ADAPT2

<u>Name</u>	<u>Location</u>	<u>Significance</u>
ADPT(1)	Local	Logical flag indicating, if true, that the interval $[x_1, x_{1+1}]$ requires adaptive integration.
ANS	Local	Current estimate of the overall integral. $ANS = I_n(a, b)$.
BIT	Local	$0.25 \Delta x_r = .25 \times DX1$.
C4D3	ADACOM	$4/3$, set by ADASET.
CURRO	TESTIN	Logical flag; if true ADAPTO is performing an outer integration.
CURR1	TESTIN	Logical flag; if true ADAPT1 is performing an outer integration.
DUM	Local	Dummy variable, used for unwanted parameter in call to MERGE.
DX	Local	$x_{1+1} - x_1$.
DX2	Local	$(x_{1+1} - x_1)/2$.
DXR	Local	$(x_{1+1} - x_1)/(b-a)$.
E(J)	ADACOM	$15/2^{J-2}$; $E(1)=15$. For use in applying equation (4.13).
EPS	ADACOM	ϵ_{rel} , relative error criterion.
EPSAB	Local	ϵ_{abs} , absolute error criterion.
EPSANT	Local	$\epsilon_{abs}(x_{1+1} - x_1)/(b-a)$, i.e. RHS of (4.32).
EPSINT	Local	$\epsilon_{rel}(x_{1+1} - x_1)/(b-a)$, for computing relative equivalent of RHS of (4.32).
ERR1	Local	$S_j^T(x, \Delta x_r) - S_j^T(x, \Delta x_r)$ for the first interval of a new subdivision.
ERR2	Local	As above for second interval.
ERREST	INFOR2	Absolute error estimate returned by the function. Used by INFINT in computing relative error.
EREMAX	Local	Maximum value of ERTRAP(1).
ERROR	Parameter	$EREMAX \times FCTR$. Returned to calling code.

Table 4.1. Continued

<u>Name</u>	<u>Location</u>	<u>Significance</u>
ERTRAP(I)	Local	$ T_3^1(x_1, x_{1+1} - x_1) - T_2^1(x_1, x_{1+1} - x_1) /3.$
EST(K,J)	Local	$S_3^{r+1}(x, x_{r+1})$ values for the r'th subdivision. (r+1-J). See figure 4.4.
ESTV(I)	Local	$S_3^1(x_1, x_{1+1} - x_1).$
ETOT	Local	Error estimate for adaptive integration over the interval $[x_1, x_{1+1}]$. (ϵ_1 in equation (4.24).)
F	Parameter	Function of X. Integrand function.
FCTR	Local	Factor for error term in multiple integration interfacing. See Section 4.1.3.
FINTG	Parameter	Five character string identifying the integrand.
FINTGO	TESTIN	Five character string identifying the integrand for ADAPTO.
FINTG1	TESTIN	Five character string identifying the integrand for ADAPT1.
I	Local	Index, identifying x_1 .
IFIN	Local	Integer, identifying error condition. See ERRORT.
IMAX	Local	Identifies interval $[x_1, x_{1+1}]$ having largest value of ERTRAP(1).
IUP	Local	Indicates sequence for treating initial intervals for further processing. IUP=0, increasing I.; IUP=1, decreasing I.
J	INFOR2	Current level of subdivision. $J = r+1$.
JOFF1	Local	Offset used for Y1. See figure 4.5.
JOFF2	Local	Offset used for Y2. See figure 4.5.
KOFF(J)	Local	Indicates which interval at the J'th level is being subdivided. KOFF(J)=0, lower interval; KOFF(J)=2, upper.
KOUNT(J)	Local	Number of intervals at the J'th level that have been processed.
LEMAX	INFOR2	Maximum level reached during adaptive integration.
MAXDIV	Local	Maximum subdivision level, as dictated by storage allocation. Currently 30.
MAXLEV	INFOR2	Maximum subdivision level, as set by user. Overwritten if greater than MAXDIV.

Table 4.1. Continued

<u>Name</u>	<u>Location</u>	<u>Significance</u>
MAXNOD	Local	Maximum number of initial subdivision nodes as dictated by storage. Currently 50.
NP	ADACH2	Number of initial subdivision nodes, as set by the algorithm.
NP1	Parameter	Number of nodes in XP1 from which initial subdivision nodes can be selected.
NVALS	INFOR2	Current number of function evaluations.
NVMAX	INFOR2	Maximum number of function evaluations for a given call to ADAPT (0, 1 or 2), as set by user.
OUTER	Local	Logical flag, if true then the function is evaluating an outer integration.
R3	ADACOM	1/3.
R12	ADACOM	1/12.
R12DX1	Local	$\Delta x_r / 12 = (x_{i+1} - x_i) / 2^{r-1} / 12.$
SUM1	Local	$S_5^r(x, \Delta x_r)$ for the lower interval at the J'th level. See figure 4.4.
SUM2	Local	$S_5^r(x, \Delta x_r)$ for the upper interval at the J'th level. See figure 4.4.
TEST	Local	1. RHS of equation (4.32), trapezoidal integrations. 2. RHS of equation (4.13), adaptive Simpson integrations.
TESTAB	Local	Integral term in (4.34), (latest estimate.
TESTEP	Local	RHS of equation (4.34).
TESTIO	TESTIN	Factor used in computing absolute error criterion from ADAPTO integration. (See Section 4.1.3.)
TESTI1	TESTIN	Factor used in computing absolute error criterion from ADAPT1 integration.
TRAP1	Local	$T_2^1(x_1, x_{i+1} - x_i).$
TRAP2	Local	$T_3^1(x_1, x_{i+1} - x_i).$
VARNO	TESTIN	5 Character string identifying the integration variable for ADAPTO.
VARN1	TESTIN	5 Character string identifying the integration variable for ADAPT1.
VARNAM	Parameter	5 Character string identifying the integration variable for the current integral.

Table 4.1. Continued

<u>Name</u>	<u>Location</u>	<u>Significance</u>
VARO1	Local	Dummy variable used to pass numerical data to ERROUT.
VARO2	Local	As above.
X1	Local	Low point of current interval being processed by the adaptive Simpson integration. $X1 = x_1$.
XCUR0	TESTIN	Current value of the integration variable for ADAPTO.
XCUR1	TESTIN	Current value of the integration variable for ADAPT1.
XINT	Local	$(b-a)$.
XL(J)	ADACOM	$1/2^{J-1}$ or $1/2^{J-2}$, used to calculate Δx_J .
XLOW(J)	Local	Current x location of the low end of the interval at the J'th level. See figure 4.4.
XMAX	Local	Internal value of b.
XMAX1	Parameter	Value of b passed into function.
XMIN	Local	Internal value of a.
XP(I)	ADACH2	Modal values for initial subdivision, as set within the function.
XP1(I)	Parameter	Modal values from which the nodes for initial subdivision can be selected.
Y(K,J)	Local	Values of the integrand function at the J'th level (4 values). See figure 4.5.
Y1	Local	Integrand value retrieved from previous level. See Figure 4.5.
Y2	Local	As above.
YHIGH(J)	Local	Integrand value at the high end of the J'th level interval, see figure 4.5.
YLOW(J)	Local	Integrand value at the low end of the J'th level interval, see figure 4.5.
YM(I)	ADACH2	Integrand value at $(x_1 + x_{1+1})/2$
YMID(J)	Local	Integrand value at the middle of the J'th level interval, see figure 4.5.
YP(I)	ADACH2	Integrand value at x_1 .

SUBROUTINE ADASET(EPS1)

Function

Subroutine ADASET initialises the constants and parameters used by the adaptive integration routines, ADAPT0, ADAPT1 and ADAPT2. These constants are either fixed for the duration of all the calculations performed by the integration routines or are such that ADASET simply defines appropriate initial values.

Parameter List

EPS1: Relative error criterion to be used by the adaptive integration algorithms.

Operation

The variables which remain fixed are stored in the COMMON block ADACOM and are given the values defined below.

XL(1): (Interval length array); 1.0

XL(J): (Interval length array; $1.0/2^{J-1}$ for J = 2 to 30.

E(1): (Error test scale array); 15.0.

E(J): (Error test scale array); $15.0/2^{J-1}$ for J = 2 to J = 30.

R3; 1/3.

R12; 1/12.

C4D3; 4/3.

EPS: (Relative error criterion); ~~that~~ stored in EPS1.

Variables that are given initial values are stored in the COMMON block TESTIN and are given the following values.

TESTI0: (Error test factor for ADAPT0); 0.0.

TESTI1: (Error test factor for ADAPT1); 0.0.

178

XCUR0: (Current value for the argument of ADAPT0); 0.0.

XCUR1: (Current value for the argument of ADAPT1); 0.0.

CURR0: (ADAPT0 active flag); false.

CURR1: (ADAPT1 active flag); false.

SUBROUTINE ERROUT (NOADPT, IFIN, FINTG, VARNAM, VARO1, VARO2, XP, YP, YM, NP)

Function

ERROUT constructs appropriate error messages according to the value of the termination flag, IFIN. The message contains information stored in the remaining parameters in the call.

Parameter List

NOADPT: The number of the adaptive integration calling ERROUT.
(e.g. for ADAPT1, NOADPT=1.)

IFIN: Termination code. (See definitions in 'operation' below.)

FINTG: 5 character text string identifying the integrand function.

VARNAM: 5 character text string identifying the integration variable.

VARO1: First numerical value to appear in the error message.

VARO2: Second numerical value to appear in the message.

XP: Array of nodal values of the integration variable.

YP: Array of nodal values of the integrand.

YM: Array of integrand values at points midway between the nodes.

NP: The number of nodes in XP.

Operation

The message consists of two parts. The first part identifies the adaptive integration routine in which the error has occurred and the status of any other routines that are currently active. The second part displays information that is relevant for the particular error that has been detected.

The first line in the first part of the error message has the form,

***** ERROR IN ADAPT* USING ***** OVER ***** #.

where the parameters NOADPT, FINTG and VARNAM are used to fill the gaps in the message appropriately. If the integration is an inner integration, two more lines may appear. These have the form,

'ADAPT*; *****; ***** = *****

and identify the status of the outer levels of integration.

For example, an error in ADAPT2 while under the control of FBET, could produce the following message.

***** ERROR IN ADAPT2 USING FALP OVER ALPHA
ADAPT1; FBET; BETA = 2500
ADAPT0; FRLT0; A = 0.56

The second part of the error message is constructed according to the type of error that has been detected and is set by the value of IFIN as described below.

(1) IFIN = 1

The space allocated in the adaptive integration routine for the storage of nodal values of the integration variable has been exceeded. (Currently 50 nodal values are provided for.) The message transmitted is,

'NODAL STORAGE LIMIT OF **** EXCEEDED'

where VAR01 provides the number of storage locations available in the adaptive integration routine. The error is fatal and the program is terminated.

#. The groups of stars, other than that at the beginning of the message indicate numbers or strings inserted at the time the message is created.

(ii) IFIN = 2

The lower limit of integration is greater than any node. Because the node sequence is used to vignette the integration interval, this condition probably amounts to an error in the code calling the adaptive integration routine.

The error message is,

'XMIN (****) GREATER THAN LARGEST NODE'

where the value of the lower limit is provided by VAR01. The error is fatal and the program is terminated.

(iii) IFIN = 3

The limit for the number of integrand function evaluations allowed for a given integration have been exceeded. The error message is

'**** FUNCTION EVALUATIONS EXCEEDED'

where the limit for the number of function evaluations is provided by VAR01.

This error message is followed by dumps of the contents of the XP, YP and YM arrays. (The format of these dumps is dictated by the subroutine ARROUT.)

The error is fatal only after 10 occurrences.

(iv) IFIN = 4

The error estimate made by the adaptive integration routine is larger than the relative error criterion unspecified. The error message is

'ERROR (*****) GREATER THAN REQUESTED (*****)'

where the error estimate and the specified error are provided by VAR01 and VAR02 respectively.

AD A145 685

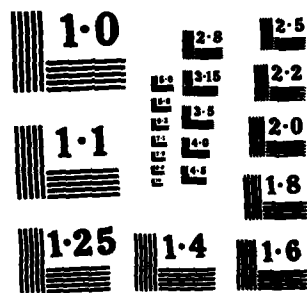
NERT : A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUN. (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARL/STRUC-397

3/7

UNCLASSIFIED

F/G 9/2

NI



The error is non fatal and control is returned to the calling code.

(v) IFIN = 5

The upper limit of integration is less than any node. This error is similar to IFIN = 2 above.

(vi) IFIN = 6

Subdivision of the integration domain has been limited. The subdivision strategy used by the adaptive integration routines is limited by either a specification for the number of subdivision levels or by the effects of roundoff errors. The error message is,

'SUBDIVISION LIMITED X = **** LEVEL = ****'

where the value of X and the level at which the subdivision stopped are provided by VAR01 and VAR02 respectively.

The error is not fatal.

4.1.3. Multiple integration - error considerations

Neglecting time dependence, the reliability functions evaluated by NERF have the most general form,

$$R = \int_{z_1}^{z_2} F_z(z) \int_{y_1(z)}^{y_2(z)} F_y(y, z) \int_{x_1(y, z)}^{x_2(y, z)} F_x(x, y, z) dx dy dz. \quad (4.38)$$

This expression can be decomposed as the following nested integration sequence;

$$R = \int_{z_1}^{z_2} I_z(z) dz, \quad (4.39)$$

$$I_z(z) = F_z(z) \int_{y_1(z)}^{y_2(z)} I_y(y, z) dy, \quad (4.40)$$

$$I_y(y, z) = F_y(y, z) \int_{x_1(y, z)}^{x_2(y, z)} F_x(x, y, z) dx. \quad (4.41)$$

The multiple integration can, in principle, be handled by a recursive sequence of calls to the same single integration algorithm. FORTRAN does not permit recursive calls to sub-programs, but the same structure can be established by having 3 copies of the same integration function, each controlling a given level of integration. Early versions of the NERF computer program did, in fact have this structure. The three functions ADAPTO, ADAPT1 and ADAPT2 were exact copies of each other and each level of integration was controlled to the same relative error criterion.

As the following argument will conclude, this approach led to a wasteful (if not accurate) integration algorithm for multiple integrals. Considerable improvements in computational efficiency could be obtained if attention was given to the interfacing between the individual single integration algorithms to form a multiple integration sequence.

Consider the outermost integration,

$$R = \int_{z_1}^{z_2} I_z(z) dz \cong I(z_1, z_2) \quad (4.42)$$

and forget, for the time being, that the evaluation of $I_z(z)$ involves further applications of the integration algorithm. Instead, regard the numerical evaluation of $I_z(z)$ as a process which returns an approximate value which is within a controlled error bound of the correct result.

Generally, the result returned by the integration algorithm is in error; $I_n(z_1, z_2)$ is returned where,

$$I_n(z_1, z_2) = I(z_1, z_2) + \epsilon_{act} \quad (4.43)$$

and ϵ_{act} is the actual error incurred during the numerical integration.

Normally, ϵ_{act} is bounded in some way by the safeguards in the algorithm,

$$|\epsilon_{act}| \leq \epsilon_{in} \quad (4.44)$$

ϵ_{in} being a numerical parameter of the algorithm which is deemed to be 'reliable' if equation (4.44) applies to every result produced by the integration. In any practical algorithm there are several sources of error:

- (i) truncation errors associated with the integration rule;
- (ii) errors in the evaluation of the integrand function;
- (iii) round-off errors caused by the finite precision of the computer, occurring at each arithmetic operation;

- (iv) non-systematic errors resulting from failure of the integration algorithm's logic to cope with unusual properties of the integrand.

Of these contributions, the first two will be assumed to be dominant. The third contribution is important if high levels of accuracy are required. Normally, the reliability functions evaluated by NERF are required to an accuracy of .01% to .1%. Round-off errors associated with eight figure operations are assumed to be at least an order of magnitude below this level. Non-systematic errors are expected to be gross in nature and detectable by the algorithm itself or obvious from discontinuities in the computed functions. Certainly this has been so for the errors that have been detected to date.

The second contribution can be represented by writing,

$$I_{zn}(z) = I_z(z) + \epsilon_z(z) \quad (4.45)$$

where I_{zn} is the integrand function returned by the numerical evaluation procedure and $\epsilon_z(z)$ is the actual error associated with a particular evaluation. Errors associated with the integration rule can be lumped together as an additive error ϵ_n , which will, in general, be sensitive to changes in $I_z(z)$ and hence will depend on the numerical estimates supplied by the integrand evaluation algorithm. This nonlinearity will be ignored so that the numerical value returned by the overall integration algorithm can be represented by,

$$I_n(z_1, z_2) = \int_{z_1}^{z_2} (I_z(z) + \epsilon_z(z)) dz + \epsilon_n. \quad (4.46)$$

The actual error is given by,

$$\epsilon_{act} = \int_{z_1}^{z_2} \epsilon_z(z) dz + \epsilon_n. \quad (4.47)$$

It is essential for the reliability of the integration procedure that,

$$\left| \int_{z_1}^{z_2} \epsilon_z(z) dz + \epsilon_n \right| \leq \epsilon_{in}. \quad (4.48)$$

Now,

$$\left| \int_{z_1}^{z_2} \epsilon_z(z) dz \right| \leq \int_{z_1}^{z_2} |\epsilon_z(z)| dz \\ \leq |\epsilon_{z,max}| \cdot |z_2 - z_1|, \quad (4.49)$$

by the mean value theorem. ($\epsilon_{z,max}$ is the maximum of $\epsilon_z(z)$ over $[z_1, z_2]$.) Without more information regarding the nature of the function $\epsilon_z(z)$, (4.49) represents an estimate for the upper bound for the magnitude of the error resulting from the evaluation of the integrand function.

Assume now, that the integration rule assures that $|\epsilon_n| \leq \epsilon_{in}$ and the integrand is evaluated within an absolute error criterion, ϵ_z governed by

$$\varepsilon_z \leq \varepsilon_{in}/|z_2 - z_1|. \quad (4.50)$$

Then,

$$\begin{aligned} \varepsilon_{act} &\leq \left| \int_{z_1}^{z_2} \varepsilon_z(z) dz + \varepsilon_n \right| \\ &\leq |\varepsilon_z| |z_2 - z_1| + |\varepsilon_n| \\ &\leq 2 \varepsilon_{in} . \end{aligned} \quad (4.51)$$

If, on the other hand, relative error criteria are used,

$$\varepsilon_n \leq \varepsilon_{rel} \int_{z_1}^{z_2} |I_z(z)| dz \quad (4.52)$$

and

$$\varepsilon_z \leq \varepsilon_{rel} |I_z(z)|, \quad (4.53)$$

then

$$\begin{aligned} \varepsilon_{act} &\leq \int_{z_1}^{z_2} |\varepsilon_z(z)| dz + \varepsilon_{rel} \int_{z_1}^{z_2} |I_z(z)| dz \\ &\leq 2 \varepsilon_{rel} \int_{z_1}^{z_2} |I_z(z)| dz. \end{aligned} \quad (4.54)$$

Equation (4.54) is the justification for using a nested^{ed} integration process where each integration is controlled by a relative error criterion equal to ε_{rel}/N where N is the number of integration levels.

If equation (4.52) is retained, but (4.53) replaced by,

$$\varepsilon_z \leq \varepsilon_{\text{rel}} \int_{z_1}^{z_2} |I_z(z)| dz / |z_2 - z_1| \quad (4.55)$$

then,

$$\begin{aligned} \varepsilon_{\text{act}} &\leq \int_{z_1}^{z_2} |I_z(z)| dz \int_{z_1}^{z_2} \varepsilon_{\text{rel}} / |z_2 - z_1| dz + \varepsilon_n \\ &\leq \int_{z_1}^{z_2} |I_z(z)| dz \varepsilon_{\text{rel}} \frac{|z_2 - z_1|}{|z_2 - z_1|} + \varepsilon_{\text{rel}} \int_{z_1}^{z_2} |I_z(z)| dz \\ &\leq 2 \varepsilon_{\text{rel}} \int_{z_1}^{z_2} |I_z(z)| dz . \end{aligned} \quad (4.56)$$

There may be significant sections of the interval $[z_1, z_2]$ for which,

$$|I_z(z)| < \int_{z_1}^{z_2} |I_z(z)| dz / |z_2 - z_1| \quad (4.57)$$

and (4.55) represents a more liberal error criterion for integrand evaluation while still satisfying the overall error criterion.

Likewise, if the inequality in (4.57) is reversed, (4.53) represents the more liberal criterion. Accordingly, the integrand evaluation criterion becomes,

$$\varepsilon_z \leq \max \left\{ \varepsilon_{\text{rel}} |I_z(z)|, \varepsilon_{\text{rel}} \int_{z_1}^{z_2} |I_z(z)| dz / |z_2 - z_1| \right\}. \quad (4.58)$$

and offers considerable gains in computational efficiency, but little loss in overall accuracy over either (4.53) or (4.55).

Returning now to the situation where $I_z(z)$ is defined by equation (4.40), ϵ_z as defined by equation (4.58) applies to the evaluation of the RHS of (4.40) which includes the function $F_z(z)$ as well as an integration over y . Assuming that the errors associated with the evaluation of $F_z(z)$ are insignificant compared with those associated with the integration then (4.58) can be satisfied if the integration algorithm is applied with ϵ_{abs} in equation (4.34) replaced by $\epsilon_{z,abs}$ where

$$\epsilon_{z,abs} = \epsilon_{rel} \int_{z_1}^{z_2} |I_z(z)| dz / |z_2 - z_1| |F_z(z)|. \quad (4.59)$$

Similarly, for the integration over x in equation (4.41), ϵ_{abs} can be replaced by $\epsilon_{y,abs}$ where

$$\epsilon_{y,abs} = \epsilon_{rel} \int_{y_1(z)}^{y_2(z)} \frac{|I_y(y,z)|}{|y_2(z) - y_1(z)| |F_y(y,z)|} dy. \quad (4.60)$$

In each case, the integral term is approximated within the algorithm which applies the latest estimate before each application of (4.34).

It is stressed here that equations (4.59) and (4.60) are applied to derive parameters which control the subdivision logic; as such they do not necessarily ensure that overall satisfaction of equation (4.44) is always achieved. For this reason, the integration algorithms must produce an estimate of ϵ_{act} . Each single integration algorithm can produce an estimate of ϵ_n for that particular level. Denoting that for the x and y levels by $\epsilon_{n,x}(y,z)$ and $\epsilon_{n,y}(z)$ respectively, the estimate for the overall error, ϵ_{est} , is

$$\epsilon_{\text{est}} = \epsilon_n + \int_{z_1}^{z_2} \left\{ F_z(z) \epsilon_{n,y}(z) + \int_{y_1(z)}^{y_2(z)} F_y(y,z) \epsilon_{n,x}(y,z) dx \right\} dy$$

.... (4.61)

Using arguments similar to those leading to equation (4.56) it can be shown that

$$\epsilon_{\text{est}} \leq 3 \epsilon_{\text{rel}} \int_{z_1}^{z_2} |F_z(z)| dz \quad (4.62)$$

provided accurate estimates of the integral terms in (4.59) and (4.60) are applied at each stage. Since this is not the case (4.62) can only be interpreted as an indication of the error magnitude. In fact the total algorithm has proved to be sufficiently conservative that there is no need to divide the required relative error by the number of levels of integration. The same error criterion is applied regardless of the number of integration levels involved.

4.1.4. Implementation (multiple integration)

Multiple integrations are handled by the set of adaptive integration sub programs, ADAPTO, ADAPT1 and ADAPT2 which have been designed to be nested (e.g. equations (4.39) to (4.41) in that order with ADAPT2 innermost. ADAPTO and ADAPT1 are similar to ADAPT2 which was described in Section 4.1.2, especially with respect to the evaluation of the integration at the level the sub program is controlling. The two outer sub programs contain additional code which provides estimates used in the application of equations (4.59) and (4.60) in ADAPT1 and ADAPT2 respectively and which performs the integrations necessary to make the error estimate defined by equation (4.61).

The code has been written in such a way that any of the integration sub programs can be omitted from the sequence, the only requirement being adherence to the order of nesting.

The sequence of operations to perform a multiple integration is illustrated schematically in Figure 4.6 which shows the flow of control between the three adaptive integration function sub programs and three integrand functions F0, F1 and F2. The diagram has been drawn according to the scheme defined in detail in Section 5.5.1. The essential elements of this scheme as they pertain to Figure 4.6, which is a simplification of similar Figures appearing in Chapter 5, are defined below.

- (1) Logic flow can be either forward (top to bottom) or reverse (bottom to top).

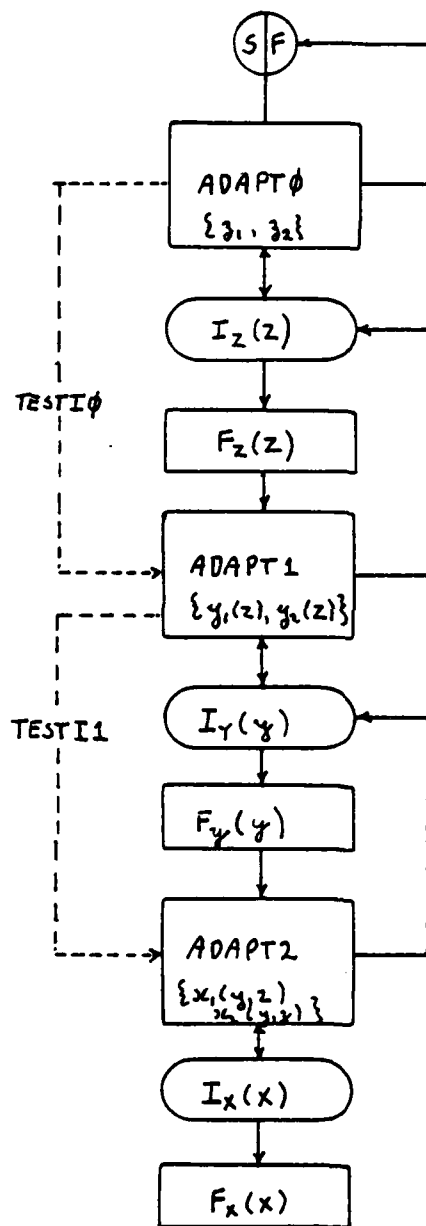


Figure 4.6. Schematic representation of the sequence of operations required to perform a triple integration. Refer to text for interpretation.

- (ii) Initial entry and final exit from the logic is at the top of the diagram.
- (iii) Small boxes indicate evaluation of the term within the box, only when the logic flow through the box is forward.
- (iv) On completion of the evaluation in the lowermost box the logic flow direction is reversed.
- (v) Large boxes indicate the adaptive integration sub programs which can direct logic flow in both directions. Integration limits are enclosed in curly brackets and are assumed to be evaluated each time the box is entered from above.
- (iv) Round boxes indicate control points associated with the evaluation of the integrand functions. The control points have memory in that reverse flow is directed along the same path as the control point was entered during the previous forward flow. The two way arrows between the control points and the adaptive integration sub programs immediately above them indicate that the control points are themselves under the control of that particular adaptive integration algorithm.

The broken lines and boxes (which are not part of the scheme used in Chapter 5) indicate the calculation of the integral estimates used for the application of equations (4.59) and (4.60).

The estimates of the integral terms in (4.59) and (4.60) are stored in TESTIO and TESTI1 in the common block TESTIN respectively.

The accumulation of the error estimates is controlled by the subroutine INFINT and associated entry points INFST, INFLE1, INFLE2 and INFSUP. For a given overall integration, INFST is called by the outermost integration control routine. This action sets the error estimates and the function evaluation counters to zero. During the integration, ADAPT1 and ADAPT2 call INFLE1 and INFLE2 respectively, at the completion of each integration at the level controlled by the particular adaptive routine. INFLE2 increments the count of function evaluations made by ADAPT2 and evaluates the term $F_y(y,z) \cdot \xi_{n,x}(y,z)$ which is, in turn, fed back via ADAPT2 to the outer integration routines. INFLE1 increments the count of function evaluations made by ADAPT1 and computes the term in curly brackets in equation (4.61) by accessing information stored in the common blocks INFOR1 and INFOR2.

On completion of the outermost integration, INFINT is called to complete the evaluation of equation (4.61) and to construct an appropriate informative message regarding the performance of the integration algorithm. The total error estimate is displayed together with that arising only from the integration of the error estimates made by the inner integration algorithms. The number function (or integrand) evaluations made at each level are also displayed.

Further details of these messages are given in the sub program descriptions which follow.

FUNCTION ADAPTO(F,XMIN1,XMAX1,XP1,NP1,FINTG,VARNAM)

Function

ADAPTO returns, as the value of the function, the integral

$$I(a,b) = \int_a^b f(x)dx.$$

The function ADAPTO is configured to operate as the outermost of a nested multiple integration sequence.

Parameter List

F: Function defining the integrand, $f(x)$.

XMIN1: Lower limit of integration, a .

XMAX1: Upper limit of integration, b .

XP1: Array containing possible boundary values of x for initial subdivisions.

NP1: Number of values in XP1. If NP1 is greater than 2 the values in XP1 must encompass XMIN1 and XMAX1. If NP1 is less than 3, ADAPTO sets up five equally spaced nodes for the initial subdivision.

FINTG: Five character string identifying the integrand.

VARNAM: Five character string identifying the integration variable.

Control Parameters Stored in Common

The following variables exercise control over the adaptive integration algorithm in ADAPTO.

NVMAX: (in INFO0) Sets the maximum number of integrand evaluations

to be used in any single integration step.

MAXLEV: (in INFORO) Sets the maximum level of subdivision (i.e. max level of r) used by the adaptive Simpson algorithm.

Operation

ADAPTO operates in the same way as ADAPT2 in: so far as the evaluation of the integral is concerned. Reference to the documentation for ADAPT2 in Section 4.1.2 should be made for details of the integration method and implementation. The variables used for the integration are defined in Table 4.1 with the only difference being that those variables listed in that Table as being located in common block INFOR2 are located, instead, in common block INFORO.

The essential difference between ADAPTO and ADAPT2 is that ADAPTO integrates the error estimate provided by the integrand function. The function F is assumed to have two arguments, the first is the value of X for which the integrand function is required. The second is the error estimate returned by the integrand function.

ADAPTO integrates the error using the same method and the intervals that are used for the integration of $f(x)$. The subdivision logic is not affected by the error integration which is not extrapolated by the Simpson algorithm. The code parallels that used for the main integration and uses the variables defined in Table 4.2.

Table 4.2.

Variables associated with error integration,
used in ADAPTO and ADAPT1 and not listed in Table 4.1.

Name	Location	Significance
ERR(K,J)	Local	Values of the integrand error at J'th level (4 values). Corresponds to Y(K,J).
ERRHIG(J)	Local	Error at high end of J'th level. Corresponds to YHIGH(J).
ERRLOW(J)	Local	Error at low end of J'th level interval. Corresponds to YLOW(J).
ERRM(I)	Local	Error at $(x_{i+1} + x_i)/2$.
ERRMID(J)	Local	Error at the middle of the J'th level interval. Corresponds to YMID(J).

FUNCTION ADAPT1(F,XMIN1,XMAX1,XP1,NP1,FINTG,VARNAM,FCTR,ERROR)

Function

ADAPT1 returns, as the value of the function, the integral

$$I(a,b) = \int_a^b f(x)dx.$$

The function ADAPT1 is configured to operate as the middle integration of a nested multiple integration sequence.

Parameter List

As for ADAPT2.

Control Parameters Stored in Common

NVMAX and MAXLEV have the same control as specified for ADAPT2, but are stored in the common block INFOR1.

Operation

ADAPT1 operates in the same way as ADAPT2 and performs the error integration described for ADAPT0. The only additional operation over those made by ADAPT0 is the application of equation (4.59) to compute an absolute error criterion.

Note that ADAPT2 sets an absolute error criterion, via equation (4.60). ADAPT0, although accommodating an absolute error criterion within the code, operates with a zero (i.e. non-operative) absolute error criterion.

SUBROUTINE INFINT(ERREQ,NOADPT,ANS)

Function

Subroutine INFINT constructs a message which indicates the error estimates and workloads associated with a given integration. The information is provided at the completion of the outermost of a nested integration sequence.

Parameter List

ERREQ: Relative error criterion used by the adaptive integration algorithm.

NOADPT: The number of the adaptive integration routine calling INFINT. (e.g. NOADPT = 1 corresponds to ADAPT1.)

ANS: The estimate of the integration as calculated by the adaptive integration algorithm.

Operation

The operation of INFINT depends on the number of the adaptive integration routine for which the information is being output. However, in all cases the main function is to assemble the information from the relevant COMMON block and to make elementary checks on the error estimates, producing an appropriate warning if suspicious integrations are detected.

(1) ADAPT2

The simplest case is when INFINT is called by ADAPT2. The integration is not nested and all the relevant information is stored in the COMMON block INFOR2. The error estimate, ERRES2 is converted to a relative error if ANS is non-zero. The informative message has the form,

'ANS = **** ERR1 = **** NVALS = ****'

where ANS, ERR1 and NVALS are the numerical value of the integration, the relative error estimate and the number of integrand evaluations used respectively.

(ii) ADAPT1

If INFINT is called by ADAPT1, then there may be an inner integration involving ADAPT2. The total error estimate consists of ERRES1 (in INFOR1) produced by the adaptive algorithm and ERRIN1 which is the total integration of the error estimates produced by ADAPT2. This total error estimate is computed by INFINT, converted to a relative error estimate and then included together with ERRIN1 in a message which has the form,

'ANS = *** ERR1 = *** ERR2 = *** NVALS = ***, ***'

where ERR1 is the total error estimate, ERR2 the estimate for the integration of the inner levels and the two numbers associated with NVALS indicate the number of integrand evaluations associated with ADAPT1 and the combination of ADAPT1 and ADAPT2 respectively.

(iii) ADAPT0

If ADAPT0 calls INFINT, the operations resemble those for ADAPT1 with the exception that the error estimates are provided by ERRES0 and ERRIN0 in INFOR0. The resulting information message is similar to that given above but with the addition of a third number after NVALS to indicate the number of evaluations associated with ADAPT0, ADAPT1 and ADAPT2.

Note that the initialisation sequence executed by INFST ensures that if any level of integration is omitted, the relevant terms contributing to total estimates are zero.

SUBROUTINE INFLE1(FACTOR,ERROR)

Function

INFLE1 produces an error estimate associated with an integration by ADAPT1. The subroutine is called by ADAPT1 only.

Parameter List

FACTOR: Term by which the error estimate is to be factored to produce the correct total estimate by the calling code. (See equation (4.61)).

ERROR: Error estimate returned by INFLE1.

Operation

Following equation (4.61) the error estimate is computed from ERRIN1 and ERRES1 the integration error estimate and the integrated error from inner levels of integration respectively.

INFLE1 also increments NV1, the total number of function evaluations associated with ADAPT1.

SUBROUTINE INFLE2(FACTOR,ERROR)

Function

INFLE2 computes the error estimate for an integration involving ADAPT2. The subroutine is called by ADAPT2 only.

Parameter List

FACTOR: Term by which the error estimate is to be factored to produce the correct contribution to the total error estimate for integrations using ADAPT2. (See equation (4.61)).

ERROR: Error estimate produced by INFLE2.

Operation

ADAPT2 is always the innermost integration and the error estimate comprises only that associated with the adaptive integration. This estimate is stored in ERRES2.

INFLE2 also accumulates the counter NV2 which indicates the total number of functions evaluations made by ADAPT2.

SUBROUTINE INFST(NOADPT,FINTG,VARNAM,XMIN,XMAX)

Function

Subroutine INFST initialises a multiple integration process by setting to zero all error estimate terms and function evaluation counters. The subroutine also constructs a prompt which identifies the integration being initialised.

Parameter List

NOADPT: The number of the integration algorithm calling INFST.

FINTG: 5 character test string identifying the integrand for the outermost integration.

VARNAM: 5 character text string identifying the integration variable for the outermost integration.

XMIN: lower limit of integration.

XMAX: Upper limit of integration.

Operation

The error estimate variables (ERRIN1, ERRINO, ERRES0, ERRES1 and ERRES2), the function evaluation counters (NV1, NV2, NVALS0, NVALS1 and NVALS2) and the test factors (TESTI0 and TESTI1) are all set to zero.

If integration information suppression is selected no further action is taken. Otherwise a prompt of the form

'ADAPT* USING **** FOR **** FROM **** to ****'

is constructed using the information in NOADPT, FINTG, VARNAM XMIN and XMAX.

204

SUBROUTINE INFSUP(LOG)

Function

Subroutine INFSUP provides the facility for optional suppression of integration information prompts. Error messages can not be suppressed.

Parameter List

LOG: Logical switch. If true all future information prompts will be suppressed. If false, all future information messages will not be suppressed.

Operation

INFSUP transfers the value of LOG to OUTSUP which is accessed by INFST and INFINT to determine whether to construct information prompts or not.

4.1.5 Performance

The error analysis presented in Section 4.1.2 and the arguments in Section 4.1.3 permit the set of appropriate convergence criteria with the adaptive integration algorithm. The analyses are not complete enough to provide accurate estimates of the errors associated with a given integration. At best an order of magnitude for the upper bound for the error can be established.

Accordingly, the integration procedure can be assessed only by appropriate tests using known integrations for which the answer can be calculated via an alternative method. There are many examples in the literature (e.g. Lyness,¹⁷ Robinson¹⁹) which demonstrate the suitability of adaptive integration methods for application for the evaluation of single integrals. In particular, the adaptive procedures are effective for dealing with improper integrals where the integrand has one or more singularities within the domain of integration.

In the present application, the integrands are not expected to be singular; the main concern is achieving adequate computational efficiency so that the cost of evaluating a multiple integration is not prohibitive.

The performance of the multiple algorithm is indicated by the results in Tables 4.3 to 4.6. The data in each table were obtained by evaluating a given integral several times using different values of ϵ_{rel} . In each case it was possible to integrate analytically so that the actual error could be evaluated. The ratio of the

estimated error to the actual error is a measure of the performance of the subdivision logic with respect to satisfying the required error bounds. This ratio must be greater than 1 and ideally as close to 1 as possible. High values indicate excessive conservatism.

The numbers NX, NY and NZ indicate the approximate numbers of integrand evaluations for each integration over the appropriate variable. NTOT indicates the total number of evaluations of the innermost integrand and, assuming an equal proportion for each integrand, is a measure of the total workload for the evaluation of the integral.

The first integration (Table 4.3) is typical of a peaked integrand with a discontinuity in one direction. The integration limits were calculated by the range limiting algorithm described in Section 4.4.1 before the integrand was truncated at $x=10$. No initial subdivision nodes were specified so that 4 intervals were used for the trapezoidal integration. Although $x=10$ was a node for the initial subdivision, the integration procedure had to find out for itself that $p_x(x) \neq 0$ over the whole of the interval $(x_3, x_4]$, ($x_3=10$). This is reflected by the relatively large numbers of function evaluations associated with the integrations in the x direction.

Note, however, that reliable results were obtained with the algorithm being more conservative at the higher levels of accuracy. This is probably caused by some interaction of numerical roundoff with the subdivision logic. Experience with the algorithm has

led to a general 'rule of thumb' that roundoff will affect the subdivision logic if the product of the number of innermost integrand evaluations and the reciprocal of the relative error requirement exceeds 10^n , where n is the number of significant figures retained by the computer. Using this rule, the 0.0001 results can be expected to be affected in this way.

If the same calculation is made with $x_2 = 10$ (Table 4.4) so that the effect of the discontinuity is moved to the integration limit, the workload is reduced by nearly an order of magnitude, with the greatest improvement being at the lowest levels of accuracy. Note that the 0.0001 result is not nearly as conservative as the corresponding result in Table 4.3.

This time, some of the lower accuracy results are conservative. This reflects the step changes in accuracy made by decisions to subdivide given intervals. Such changes can result in integrations involving few function evaluations being excessively conservative.

The results in Table 4.5. indicate adequate performances for an integration which involves further integrand functions that are typical of those used to represent the density functions used in reliability models for fatigue.

An integration involving variable integration limits is given in Table 4.6. Note that the innermost integration is trivial and the algorithm has reacted accordingly. Again note the extreme conservatism at low levels of accuracy: the minimum

number of integrand evaluations programmed into the subdivision logic is more than adequate for the 0.1 integration.

The results tabulated here are typical of those that have been obtained for a wide range of integrations. They are not conclusive evidence of accuracy; such tests never can be. They do however indicate that the multiple integration algorithm can work most effectively for ϵ_{rel} in the range 0.01 to 0.0001 on an eight figure computer. This is more than adequate for the requirements of reliability functions based on input data exhibiting much greater sources of error.

For these levels of accuracy approximately 25-30 integrand evaluations can be expected for each integration over one of the variables of integration. Numbers around 100 indicate a discontinuity in the integrand which should be removed by redefining the limits of integration. (This accounts for the attention given to integration limits in the reliability modelling.) Larger numbers of integrand evaluations indicate, either several discontinuities in the integrand function, or some other problem which causes an excessive demand for interval subdivision.

ϵ_{rel}	Answer returned	ϵ_{est} (rel.)	$\epsilon_{est}/\epsilon_{act}$	NZ	NY	NX	NTOT
.1	.5017	.00313	0.92	21	19	87	35583
.01	.50133	.00301	1.13	21	21	103	45423
.001	.5000048	.000093	9.6	33	27	115	101315
.0001	.5000037	.000092	12.4	33	28	143	132847

Table 4.3. Performance statistics for,

$$I = \int_{z_1}^{z_2} \int_{y_1}^{y_2} \int_{x_1}^{x_2} p(x)p(y)p(z) dx dy dz$$

for $p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2$ and $p(y)$ and $p(z)$ similarly defined.
 $\mu=10, \sigma=0.1$. $x_1=y_1=z_1=-9.371$ and $z_2=y_2=x_2=10.631$ (limits
 calculated by RANGE, Section 4.4.1). $p(x)=0$ for $x > 10$.

Maximum subdivision level 20.

ϵ_{rel}	Answer returned	ϵ_{est} (rel.)	$\epsilon_{est}/\epsilon_{act}$	NZ	NY	NX	NTOT
.1	.5017	.0405	15.5	21	19	9	3681
.01	.50145	.0037	1.05	21	21	13	5733
.001	.5000045	.000105	11.6	33	27	19	16739
.0001	.500016	.0000865	2.7	33	28	27	25083

Table 4.4. Performance statistics for integration defined
 for table 4.3 but with $x_2=10$.

ϵ_{rel}	Answer returned	ϵ_{est} (rel.)	$\epsilon_{est}/\epsilon_{act}$	NZ	NY	NX	NTOT
.1	8.1527	.0553	2.89	15	15	15	3375
.01	8.00273	.0011	3.2	25	23	23	14325
.001	8.000755	.000823	8.72	29	28	29	23229
.0001	8.000024	.0000171	5.7	59	50	59	174463

Table 4.5. Performance statistics for the integration,

$$I = \int_0^{\infty} \int_0^{\infty} \int_0^{\infty} z^n e^{-z} \int_0^{\infty} y^n e^{-y} \int_0^{\infty} x^n e^{-x} dx dy dz$$

$$= (n!)^3; = 8 \text{ for } n=2.$$

ϵ_{rel}	Answer returned	ϵ_{est} (rel.)	$\epsilon_{est}/\epsilon_{act}$	NZ	NY	NX	NTOT
.1	192.7542	.0784	31	9	8	9	549
.01	192.1887	.00282	6.33	25	14	9	2907
.001	192.2677	.000068	1.98	33	19	9	5283
.0001	192.2740	.000039	24.9	33	25	9	7155
.00001	192.2743	.000004		65	33	9	18693

Table 4.6. Performance statistics for the integration,

$$I = \int_{-\pi/2}^{\pi/2} \int_0^2 \int_0^2 2a \cos \theta (4a^2 - r^2)^{1/2} r^3 dz dr d\theta$$

$$= 64a^5(\pi - 26/15)/15; = 192.2743 \text{ for } a=2.$$

4.2. Interpolation

One of the significant features of the NERF computer program is that input functions can be defined as sets of ordered pairs of argument and function values. This removes a requirement for the user to fit specific functional forms to the input data and removes any dependency on such forms from the NERF code.

The generation of continuous functions from these sequences of ordered pairs requires an appropriate interpolation procedure. For several years during the development of NERF, piecewise polynomial interpolation was used. As the integration methods were refined and the graphical methods for the display of the adaptive integration process developed, it became evident that discontinuities in the derivatives of the interpolated functions were causing an interference with the adaptive integration logic resulting in excessive numbers of function evaluations in some cases. Although this problem could, to a large extent, be overcome by locking the adaptive integration logic to the nodes used to define the input functions, the use of splines which ensure continuity of at least some of the derivatives proved beneficial.

Although the algorithm used to fit piecewise polynomials to the ordered pairs could cope with polynomials of any order, experience showed that polynomials of order 3 and 5 produced the most satisfactory results especially when the interpolated function was inspected graphically. Accordingly, cubic splines were adopted as a replacement for the piecewise polynomial interpolation procedure.

4.2.1. Mathematical Basis

The theoretical basis for spline interpolation is well known and documented by most good texts on numerical analysis (e.g. Ahlberg et al ref.21). A cubic spline is the simplest spline interpolation function and has a physical analogue in the draftsman's spline (from which the name originated) which consists of thin strips of wood, or similar material, used to fit smooth curves through points on a diagram by attaching lead weights at points along the strip and allowing the elasticity of the wood and the effect of gravity to generate the required distribution of curvature.

Given a sequence $\{x_i\}$ of M argument values for the function $f(x)$ the cubic spline through the points $\{x_i, f(x_i)\}$ is identical to the curve that the draftsman would obtain if the points at which the lead weights were attached coincide with the points $\{x_i, f(x_i)\}$. The cubic spline has continuous first and second derivatives and of all the functions having a continuous second derivative over the interval $[x_1, x_M]$ the cubic spline minimises the integral,

$$\int_{x_1}^{x_M} f''(x)^2 dx.$$

and therefore has minimum curvature.

(4.63)

The cubic spline consists of a sequence of 3rd order polynomials, one for each of the intervals $[x_i, x_{i+1}]$ for $i=1, M-1$. Denoting the cubic spline by $y(x)$, the following notation may be defined.

$$s(x) = dy/dx \quad (4.64)$$

$$m(x) = d^2y/dx^2 \quad (4.65)$$

$$y_i = y(x_i) \quad (4.66)$$

$$s_i = s(x_i) \quad (4.67)$$

$$m_1 = m(x_1). \quad (4.68)$$

The equation defining the spline over $[x_1, x_{i+1}]$ is generated by assuming that the second derivative varies linearly over the the interval, i.e.,

$$m(x) = m_1 + (m_{i+1} - m_1)(x - x_1)/h_1, \quad (4.69)$$

where

$$h_1 = x_{i+1} - x_1. \quad (4.70)$$

Integrating equation (4.69) yields,

$$s(x) = m_1(x - x_1) + (m_{i+1} - m_1)(x - x_1)^2/(2h_1) + A_1 \quad (4.71)$$

$$y(x) = m_1(x - x_1)^2 + (m_{i+1} - m_1)(x - x_1)^3/(6h_1) + A_1(x - x_1) + B_1, \quad (4.72)$$

where A_1 and B_1 are constants of integration which may be evaluated by imposing the conditions $y(x_1) = y_1$ and $y(x_{i+1}) = y_{i+1}$, so that eventually

$$y(x) = y_1 + (y_{i+1} - y_1)(x - x_1)/h_1 - h_1(2m_1 + m_{i+1})(x - x_1)/6 + m_1(x - x_1)^2/2 + (m_{i+1} - m_1)(x - x_1)^3/6, \quad (4.73)$$

and

$$s(x) = m_1(x - x_1) + (m_{i+1} - m_1)(x - x_1)^2/(2h_1) + (y_{i+1} - y_1)/h_1 - h_1(2m_1 + m_{i+1})/6. \quad (4.74)$$

The second derivatives at $x = x_1$ and $x = x_{i+1}$, m_1 and m_{i+1} respectively are, at this stage, unspecified. Equations (4.73) and (4.74) are applicable for any cubic passing through y_1 and y_{i+1} . The special condition leading to the cubic spline is that the first derivative is continuous. Enforcement of this condition leads to a set of equations that can be solved to yield the point values of curvature, $\{m_i\}$. From (4.74)

2/4

$$s_i = s(x_i) = (y_{i+1} - y_i)/h_i - h_i(2m_i + m_{i+1})/6 \quad (4.75)$$

$$s_{i+1} = s(x_{i+1}) = (y_{i+1} - y_i)/h_i + h_i(m_i + 2m_{i+1})/6. \quad (4.76)$$

Equation (4.76) for the interval $[x_{i-1}, x_i]$ yields,

$$s_i = (y_i - y_{i-1})/h_i + h_{i-1}(m_{i-1} + 2m_i)/6. \quad (4.77)$$

Combining equations (4.75) and (4.77),

$$\begin{aligned} m_{i-1}h_{i-1}/(h_i + h_{i-1}) + 2m_i + m_{i+1}h_i/(h_i + h_{i-1}) \\ = 6[(y_{i+1} - y_i)/h_i - (y_i - y_{i-1})/h_{i-1}]/(h_i + h_{i-1}), \end{aligned} \quad (4.78)$$

which is valid for $i=2, M-1$. The two extra equations required to completely specify $\{m_i\}$ are obtained by imposing conditions on the slope of the spline at each end of the interval $[x_1, x_M]$. Considering the end, $x=x_1$, and denoting the imposed slope by y'_1 , equation (4.75) yields

$$y'_1 = (y_2 - y_1)/h_1 - h_1(2m_1 + m_2)/6 \quad (4.79)$$

which can be written in the form

$$2m_1 + m_2 = d_1 \quad (4.80)$$

where

$$d_1 = 6[(y_2 - y_1)/h_1 - y'_1]/h_1. \quad (4.81)$$

Similarly, from (4.77) at $i=M$,

$$m_{M-1} + 2m_M = d_M, \quad (4.82)$$

where

$$d_M = -6[(y_M - y_{M-1})/h_M - y'_M]/h_M. \quad (4.83)$$

The specification for y'_1 and y'_M is arbitrary and may be adjusted to meet specific requirements. The specification in NERF amounts to a second order estimation of the slope using the 3 data pairs at each end of the spline. For example, at $x=x_1$, Taylor's expansions about $x=x_1$, yield

$$(y_2 - y_1)/h_1 = y'_1 + y''_1 h_1/2 + o(h_1^2) \quad (4.84)$$

$$(y_3 - y_1)(h_1 + h_2) = y'_1 + y''_1 (h_1 + h_2)/2 + o((h_1 + h_2)/2). \quad (4.85)$$

Eliminating y''_1 from these equations yields,

$$d_1 = 6[(y_3 - y_1)/(h_1 + h_2) - (y_2 - y_1)h_1]/h_2. \quad (4.86)$$

Similarly, expansion about $x=x_M$ leads to

$$d_M = -6[(y_M - y_{M-2})/(x_M - x_{M-2}) - (y_M - y_{M-1})/h_{M-1}]/h_{M-1} \quad (4.87)$$

The set of equations (4.78), (4.80) and (4.82) can be solved to yield the curvatures $\{m_i\}$ and the cubic spline is completely determined.

4.2.2. Implementation

Given the sequences $\{x_i\}$ and $\{y_i\}$, the construction of a cubic spline consists of solving equations (4.78), (4.80) and () for m_i . These solutions are obtained during an initialisation phase and stored, together with the first order differences,

$$y_i^D = (y_{i+1} - y_i) / h_i \quad (4.87)$$

for later use by equations (4.71) or (4.72) for evaluation of $y(x)$ or $s(x)$. Slightly more efficient forms of equations (4.71) and () can be generated by rearrangement, i.e.,

$$y(x) = y_i + (x - x_i) y_i^D + (x - x_i)(x - x_{i+1})(m_i + m_{i+1} + m(x)) / 6 \quad (4.88)$$

$$s(x) = y_i^D + (2x - x_i - x_{i+1})(m_i + m_{i+1} + m(x)) + (x - x_i)(x - x_{i+1})(m_{i+1} - m_i) / (6h_i), \quad (4.89)$$

where $m(x)$ is given by (4.69).

The main task during initialisation is the solution of the set of equations defining $\{m_i\}$. These equations can be recast in the form,

$$2m_1 + \lambda_1 m_2 = d_1 \quad (4.90)$$

$$(1 - \gamma_i) m_{i-1} + 2m_i + \gamma_i m_{i+1} = d_i \quad (4.91)$$

$$\lambda_M m_{M-1} + 2m_M = d_M, \quad (4.92)$$

where

$$\gamma_i = h_i / (h_i + h_{i+1}) \quad (4.93)$$

$$d_i = 6(y_i^D - y_{i-1}^D) / (h_i + h_{i+1}), \quad (4.94)$$

and

$$\lambda_1 = \lambda_M = 1. \quad (4.95)$$

(λ_1 and λ_M are retained as separate parameters to allow possible modifications to the code to cope with more general boundary conditions as described by (ref)).

The set of equations thus form a 'tridiagonal' set which may be solved by the algorithm usually attributed to Thomas (ref), the form appropriate for the equations here being described below.

$$(i) \text{ Set } b_1 = 2, a_1 = d_1/2 \text{ and } c_1 = \lambda_1/b_1. \quad (4.96)$$

(ii) For $i=2$ to $i=M-1$,

$$b_i = 2 - (1 - \gamma_i) c_{i-1}, \quad (4.97)$$

$$c_i = \gamma_i / b_i, \quad (4.98)$$

$$a_i = (d_i - (1 - \gamma_i) a_{i-1}) / b_i. \quad (4.99)$$

$$(iii) \text{ Set } b_M = 2 - \lambda_M c_{M-1} \quad (4.100)$$

$$\text{and } a_M = d_M - \lambda_M a_{M-1}. \quad (4.101)$$

(iv) For $i = M-1$ to $i=1$,

$$a_i = a_i - c_i a_{i+1}. \quad (4.102)$$

Following these operations the values $\{a_i\}$ are equal to the required solutions for $\{m_i\}$.

The spline interpolation is provided by the function FINTRP which executes the tasks of initialisation and interpolation according to the value of a control parameter in the call. The evaluation of the derivative function is provided by an entry point in FINTRP called DERIV.

4.2.3. Inverse interpolation

Given a function defined by a set of ordered pairs of function and argument and evaluated by an interpolation procedure, an obvious approach for generating the inverse function (e.g. $x=x(y)$ for $y(x)$) is to use direct interpolation with the sequences $\{x_i\}$ and $\{y_i\}$ interchanged. Unfortunately, the inverse function thus generated is not an exact reciprocal of the forward interpolation. In other words, if $y^I(x)$ denotes the interpolated value of $y(x)$ using $\{x_i\}$ and $\{y_i\}$, and $x^I(y)$ the interpolated value of $x(y)$ using $\{y_i\}$ and $\{x_i\}$, then

$$x^I(y^I) \neq x. \quad (4.103)$$

This non-reciprocity (which is not precisely known) can lead to hysteresis if repeated evaluations of the interpolated function and the inverse are made. Such hysteresis contributed to problems during the development of an interpolation procedure for the loss factor where the evaluation of ψ^{-1} is required.

An alternative approach is to obtain the inverse by solving

$$y^I(x) - y = 0 \quad (4.104)$$

using the secant method described in Section 4.3. By successive iterations, the error in (4.103) and hence the non-reciprocity can be controlled to known limits. Any sensitive logic can then be modified to cope.

FUNCTION FINTRP(X,XAR,A,IMAX,ITYPE)

Function

FINTRP evaluates the interpolating function for the set of ordered pairs of argument and function values stored in XAR and A respectively. The current implementation uses cubic spline interpolation.

Parameter List

X: Value of the argument for which the interpolated function is required.

XAR: Array containing $\{x_i\}$, the values of argument used to define the function.

A: Array containing $\{y_i\}$, the function values used to define the function. Note that A must be large enough to store the sequences $\{y_i^D\}$ and $\{m_i\}$.

IMAX: Number of values in each of $\{x_i\}$ and $\{y_i\}$.

ITYPE: Control parameter.

=1, or =2, The interpolating function is evaluated using values for $\{y_i^D\}$ and $\{m_i\}$ that are assumed to be stored in A.

=3, The spline is initialised and the interpolated value for $x=X$ evaluated.

=4, The spline is initialised but no evaluation of the interpolating function is made.

Operation

The array A is assumed to be sufficiently large to store $\{y_i\}$, $\{y_i^D\}$ and $\{m_i\}$ for $i = 1$ to IMAX. During initialisation, the first order differences are computed via equation (A.37) and the curvature values are computed using the algorithm described in Section 4.4.2. The code is a direct application

of this algorithm with the following equivalences between mathematical symbols and variable names applicable.

$$ALO = \lambda_1,$$

$$ALM = \lambda_M,$$

$$DO = d_1,$$

$$DM = d_M,$$

$$B(I) = b_1,$$

$$C(I) = c_1,$$

$$A(I + 2*IMAX) = a_1 = m_1,$$

$$HI = h_1,$$

$$HIM = h_{i-1},$$

$$HT = h_1 + h_{i+1}.$$

The evaluation of the spline function at $x=X$ follows the same logic as that for evaluating the derivative. DERIV is, in fact, an entry point which sets ITYPE = 5. The following operations complete the evaluation of the interpolating function, or the derivative according to the value of ITYPE.

- (i) INDLOW is called to find the interval containing X. (i=IX).
- (ii) If X is outside $[x_1, x_{IMAX}]$ a warning prompt is issued.
- (iii) If X is close to one of the values of $\{x_i\}$, the corresponding value of $\{y_i\}$ is returned rather than perform a full interpolation. Note that EPS1 is set to a number corresponding to machine precision. If the

derivative is required, equations (4.75) or (4.76) are used to return a nodal value.

- (iv) Equations (4.88) or (4.89) are evaluated for the interpolating function or the derivative respectively. Note the following equivalences between the mathematical notation and the program variable names.

$$\text{BITX} = x - x_1,$$

$$\text{RMI} = m_1,$$

$$\text{RMIP1} = m_{i+1},$$

$$\text{SM} = (m_{i+1} - m_1) / h_1,$$

$$\text{RM} = m(x).$$

Prompts

- (i) 'CAUTION EXTRAPOLATION X = *****': The value of x is outside the interval $[x_1, x_M]$ and the interpolation is probably being called erroneously.

FUNCTION DERIV(X,XAR,A,IMAX)

Function

DERIV evaluates the derivative of a function defined by a sequence of ordered pairs.

Parameter List

X: Value of x for which the derivative is required.

XAR: Sequence of values of argument defining the function.

A: Sequence of function values defining the function.

IMAX: Number of data values in XAR or A.

Operation

DERIV is an entry point in FINRP and the operations required for the evaluation of the derivative are described with the documentation for FINTRP.

4.3. Solution of Equations

The limit risk inspection procedure requires the solution of

$$\log(r(t)) = \log(r_{lim}) \quad (4.105)$$

for t . The function $r(t)$ is given by the appropriate expression for total risk and although defined for any value of t , its inverse is intractable by analytical methods. Numerical solution of (4.105) is necessitated. The secant method (ref 22) is suitably efficient and easy to apply. Moreover, calculation of the derivative of $\log(r(t))$ is not required.

Although the calculation of inspection times using the limit risk criterion was the motivation for the provision of an equation solver within the NERF program, the secant method is also used to provide inverse interpolation for the input functions that are described by ordered pairs and to solve the equations associated with the proof load inspection boundary (Section 5.6.).

4.3.1. The secant method

Consider the solution of the equation

$$f(x)=0. \quad (4.106)$$

Given two initial estimates for x , x_1 and x_2 , and corresponding function values, $f(x_1)$ and $f(x_2)$, an estimate can be found by inverse linear interpolation, i.e.,

$$x_3 = x_1 - (x_2 - x_1) \cdot f(x_1) / (f(x_2) - f(x_1)) \quad (4.107)$$

as shown schematically in Figure 4.7 . The values of x_2 and x_3 can be similarly used to yield a new estimate for the solution of equation (4.106), leading to the recurrence relationship,

$$x_{i+1} = x_i - (x_i - x_{i-1}) \cdot f(x_{i-1}) / (f(x_i) - f(x_{i-1})) \quad (4.108)$$

which is the basis of the secant method. Equation (4.108) can be applied repeatedly until a suitable convergence criterion has been satisfied. In NERF, the criterion,

$$|f(x)| < \epsilon \quad (4.109)$$

is used. The sequences of iterates produced by equation (4.108) can be shown (ref 22) to have good rates of convergence.

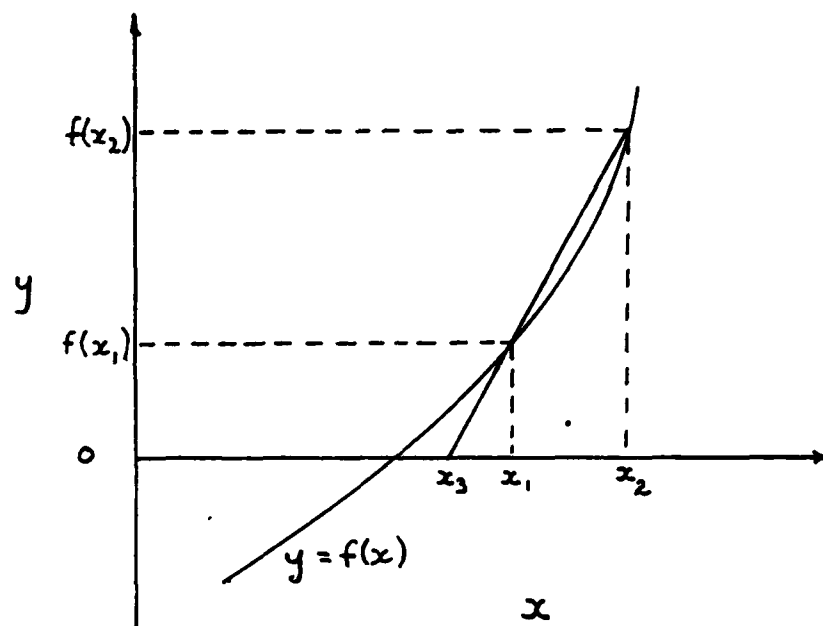


Figure 4.7. Geometric basis for the secant method.

4.3.2. Implementation

The solution of equation (4.106) is controlled by the subroutine FSOLVE which returns the solution as the value of the function. A direct copy, FSOLV2, permits the solution procedure to be nested.

The implementation of the secant method in NERF has the refinement that, following each application of equation (4.108), it is not necessarily true that x_{i-1} is the estimate that is discarded. Instead, the larger of x_i and x_{i-1} is discarded. This refinement allows the initial estimates to be defined in any order and can improve initial rates of convergence.

Typically, adequate convergence (to a relative error of 10^{-5} say) can be achieved in less than 10 iterations. The code in FSOLVE flags an iterative sequence lasting more than 20 iterations as a possible error condition.

```
FUNCTION FSOLVE(F,ARG,GIN1,GIN2,FV1,FV2,EPS)
```

```
FUNCTION FSLOV2(F,ARG,GIN1,GIN2,FV1,FV2,EPS)
```

Function

FSOLVE, and its direct copy FSOLV2 solves the equation

$$F(x) - ARG = 0$$

using the secant method.

Parameter List

F: Function defining $F(x)$.

ARG: Value of $F(x)$ for which the solution is required.

GIN1: First estimate of x , x_1 .

GIN2: Second estimate of x , x_2 .

FV1: $F(x_1)$.

FV2: $F(x_2)$.

EPS: Absolute convergence criterion.

Operation

The code in FSOLVE is a direct application of the implementation described in Section 4.3.2. Note that the added acceptance criterion

$$f(x_{i+1}) - f(x_i) < \epsilon$$

where ϵ is a very small number (e.g. 10^{-32}) is also applied to guard against the effects of roundoff errors.

Error Messages

- (1) 'CONVERGENCE FAILURE IN FSOLVE': More than 20 iterations have been used. FSOLVE returns the current estimate to the calling code.

4.4. Miscellaneous Support Functions

In addition to the three basic numerical methods of integration, solution of equations and interpolation, the sections of code that evaluate the reliability functions depend on the availability of subroutines which perform certain support functions. These subroutines are described in this Section.

4.4.1. Function range limiting

Some of the input functions accesses by NERF are defined for infinite ranges of the argument. It is important, in so far as the numerical routines are concerned, that the infinite range be converted to a finite range, beyond which the function is sensibly zero.

This conversion is provided by the subroutine RANGE which, given the range of the arguments for a function, computes a new range if possible such that the function is greater than a small number (currently 10^{-8}) over the whole of the new range. The subroutine also finds the location of the maximum value of the function. (This additional facility is not used by the reliability function evaluation code.)

SUBROUTINE RANGE(F,XLOW,XHIGH,XPEAK,IPEAK)

Function

Given a function, $F(x)$, RANGE finds the limits for the argument, x , beyond which $F(x)$ is insignificantly small. RANGE also locates the maximum value of $F(x)$.

Parameter List

F: Function for which the argument limits are required.

XLOW: Specified lower limit for x from which the search for the 'sensible' limit will commence. Returned as the located sensible limit.

XHIGH: Specified upper limit for x , from which the search for the sensible limit will commence. Returned as the sensible limit.

XPEAK: Returned as the argument corresponding to the maximum value of $F(x)$.

IPEAK: Control parameter.

=0, RANGE does not locate the maximum value of $F(x)$.

≠0, RANGE locates the maximum value of $F(x)$.

Operation

The search for the limits and function maximum uses a conventional interval bisection method. The search is controlled by three parameters which are given values RANGE.EPS specifies the minimum value of the function used to find the sensible limits. Currently EPS is set to 10^{-8} . NLIM and NLOW specify the numbers of interval subdivisions used to locate the limits and maximum respectively. Currently, NZERO = 10 and NLIM = 20.

Prompts

- (i) '*** WARNING ** F(XHIGH) = ****': The value of $F(x)$ for the upper sensible limit is greater than EPS. This means that either the search has failed or that the upper limit specified during the call vignettted the function. The latter case may be legitimate.
- (ii) '*** WARNING ** F(XLOW) = ****': The value of $F(x)$ at the lower sensible limit is greater than EPS. See (i) above.

4.4.2. Index Location

Given a monotonic sequence, $\{x_i\}$, of nodal values of a variable, x , an ordering of the index i is defined. A requirement of many of the numerical procedures used by NERF is to locate a given value of x , x_{val} , in the sequence.

Defining the notion that the direction of i increasing is movement from left to right (as would be the case if the sequence of values were written across a page), function INDHI finds the highest value of i such that x_i is to the left of x_{val} . Function INDLOW finds the lowest value of i such that x_i is to the right of x_{val} .

A refinement of this process can be made by allowing the specification of a tolerance for the search. If ϵ is such a tolerance, then INDHI finds the highest value of i such that x_i is to the left of the interval $(x_{val} - \epsilon, x_{val} + \epsilon)$. INDLOW finds the lowest value of i such that x_i is to the right of the interval $(x_{val} - \epsilon, x_{val} + \epsilon)$.

The operation of these two functions is described below.

FUNCTION INDHI(XVAL,X,N,EPS)

Function

Given that the array X contains a sequence $\{x_i\}$ of monotonic increasing or decreasing values of x, then INDHI finds the highest value of i such that x_i is to the right of x_{val} , (see Section 4.4.2.). An error margin, ϵ , is allowed for so that for $\{x_i\}$ increasing,

$$INDHI = \max\{i : x_i < x_{val} - \epsilon\}$$

and for $\{x_i\}$ decreasing,

$$INDHI = \max\{i : x_i > x_{val} + \epsilon\}.$$

Parameter List

XVAL: Value of x for which the index is required.

X: Array containing $\{x_i\}$.

N: Number of values in $\{x_i\}$.

EPS: Error tolerance, ϵ .

Operation

For $\{x_i\}$ both increasing and decreasing, the required index is located by searching from $i=N$ to $i=1$ until the first node is found that meets either $x_i < x_{val} - \epsilon$, ($\{x_i\}$ increasing), or $x_i > x_{val} + \epsilon$, ($\{x_i\}$ decreasing).

Note that if all the values in $\{x_i\}$ are greater than x, INDHI returns the value 0.

FUNCTION INDLOW(XVAL,X,N,EPS)

Function

Given that the array X contains a sequence $\{x_i\}$ of monotonic increasing or decreasing values of x, then INDLOW finds the lowest value of i such that x_i is to the left of x_{val} , (see Section 4.4.2.). An error margin, ϵ , is allowed for so that for $\{x_i\}$ increasing,

$$\text{INDLOW} = \min\{i : x_i > x_{val} + \epsilon\}$$

or for $\{x_i\}$ decreasing,

$$\text{INDLOW} = \min\{i : x_i < x_{val} - \epsilon\}$$

Parameter List

XVAL: Value of x for which the index is required.

X: Array containing $\{x_i\}$.

N: Number of values in $\{x_i\}$.

EPS: Error tolerance, .

Operation

For $\{x_i\}$ both increasing and decreasing, the required index is located by searching from $i=1$ to $i=N$ until the first node is found that meets either $x_i > x_{val} + \epsilon$, ($\{x_i\}$ increasing), or $x_i < x_{val} - \epsilon$, ($\{x_i\}$ decreasing).

Note that if all the values of $\{x_i\}$ are less than x, INDLOW returns the Value N+1.

4.4.3. Merging

Given two monotonic increasing sequences, $\{x_i\}$ and $\{x_j\}$ say, of a variable, x say, the operation of merging these sequences to form a new sequence $\{x_k\}$ can be defined in the following way.

The new sequence contains a strictly monotonic sequence of values of x selected from $\{x_i\}$ and $\{x_j\}$. Values of $\{x_i\}$ and $\{x_j\}$ that are coincident, (either within the original sequences or between sequences), within a 'zone of coincidence', \mathcal{E} , are coalesced as a single value in $\{x_k\}$. The sequence $\{x_k\}$ can be vignettted so that values within the interval $[x_{\min}, x_{\max}]$ only are included. If no values in $\{x_i\}$ or $\{x_j\}$ are coincident within x_{\min} and x_{\max} (within \mathcal{E}) then the limiting values are inserted in the sequence $\{x_k\}$.

In all cases the smallest value of x in the sequence $\{x_k\}$ is that with $k=1$.

If one of the sequences $\{x_i\}$ or $\{x_j\}$ is empty, then the operation of merging reduces to a vignetting operation to produce a sequence confined to the interval $[x_{\min}, x_{\max}]$.

Subroutine MERGE, described below, performs the operation of merging.

SUBROUTINE MERGE(X1,M1,X2,M2,OUT,XLOW,XHIGH,EPS)

Function

Given two monotonic increasing sequences of values of x , ($\{x_1\}$ and $\{x_j\}$), stored in the arrays X1 and X2, MERGE forms a new monotonic increasing sequence of values, $\{x_k\}$, according to the definitions given in Section 4.4.3.

Parameter List

X1: Array containing $\{x_1\}$.

M1: Number of values in $\{x_1\}$.

X2: Array containing $\{x_j\}$.

M2: number of values in $\{x_j\}$.

OUT: Array in which the new sequence will be returned to the calling code.

M: On entry; number of values OUT can store. Returned as the number of values placed in OUT.

XLOW: Lower limit for x in the new sequence.

XHIGH: Upper limit for x in the new sequence.

EPS: Coincidence margin.

Significant local variables

EMPTY1: Logical switch signifying, when true, that $\{x_1\}$ is empty.

EMPTY2: Logical switch signifying, when true, that $\{x_j\}$ is empty.

J: Counter for OUT. Location of last entered value.

I1: Counter for X1. Location of next value to be considered for placement in the new sequence.

236

I2: Counter for X2. Location of next value to be considered for placement in new sequence.

L1LAST: Location of last acceptable value in X1.

L2LAST: Location of last acceptable value in X2.

CURR: Last value placed in the new sequence.

Error Messages

(1) 'ERROR IN MERGE ... XLOW IS GREATER THAN XHIGH'

'XLOW = ***** XHIGH = *****'

5. DESCRIPTION OF THE NERF COMPUTER CODE

The previous Chapter described the various numerical methods and associated FORTRAN subroutines and functions which, although part of NERF may, in principle, be used separately. These routines were capable of description as separate entities. The remainder of the routines which comprise the NERF computer program perform operations that are more specifically related to the evaluation of reliability functions: their descriptions are more effectively made when considered as part of the overall computer program.

5.1. General Coding Philosophies and Methods

NERF was designed primarily to evaluate the reliability functions derived in Chapter 3. A secondary, and vitally important objective was to provide versatile methods for data input, result presentation and fault diagnosis. It is therefore a large program but is highly modular, containing over 100 FORTRAN subroutines or functions, each of which performs a well defined and specific task.

The construction of a computer program of this size requires careful thought and the subsequent documentation more so. Generally, the code was written in such a way that a balance between computational efficiency and code 'transparency' was achieved. To a large extent, 'transparency' was assured by the high degree of modularity; the majority of subroutines contain less than one page of coding and each significant operation within a routine is separated by appropriate comments so that, given a statement of a routine's operation, an experienced FORTRAN programmer should have little difficulty establishing 'what the code is doing'.

The subroutines in NERF form a hierarchy typical of any large computer program. At the apex of the hierarchy, the 'main program' controls the overall sequence of operations. At

the base of the hierarchy there are many subroutines which perform basic numerical or communications tasks. The bulk of the routines lie between these two extremes. The higher a subroutine or function is in the hierarchy the more global is its operation.

The documentation presented in this Chapter concentrates most heavily on the higher level subroutines. Each routine is described, at least, by a stylised summary as developed in the previous Chapter. For many low level routines, this summary together with the code itself is sufficient documentation. Higher level routines, because of the more general and often complex interaction with other routines may require flow diagrams and, especially in the case of the mathematical functions used to evaluate the reliability functions, detailed mathematical descriptions.

In these descriptions and flow diagrams, a mixed notation comprising both mathematical symbols and program variable names is used. Often the mixed notation will provide implicit definition of local variables in a routine: because most routines are short, explicit definition of all local variables is not given.

Variables in COMMON storage are, however, described in detail in Appendix A.2. COMMON storage is structured, consisting of many named COMMON blocks each of which contains a small number of related variables. It is suggested that the reader becomes familiar with the variable names before attempting to understand the detailed documentation presented in this Chapter.

The current version of NERF is the result of several years of development of computer programs for the evaluation of reliability functions and this development has spanned several changes in mathematical notation. Consequently, the names of many of the variables do not bear an obvious relationship with the mathematical entities they represent. Thus far it has

239

been considered impractical to undertake the large task of changing the variables names throughout the code.

5.2. Principal Phases of Operation and Program Output

The sequence of general operations performed by NERF can be divided into five phases; Initialisation, data input and initialisation of the computational algorithms, the construction of a time sequence of reliability functions, the calculation of strength distributions and, finally, program termination.

This sequence of operations is controlled by the 'main program', called PROGRAM NERF, which resides at the apex of the hierarchy of subroutines in NERF. The control of major operations is effected by calls to appropriate subroutines, the description of which forms the bulk of the remainder of this Section.

The first phase, 'initialisation', is short and is executed by code within PROGRAM NERF. The run time clock is initialised by calling a system supplied subroutine, TIMES, which computes the current job time in milliseconds. The various output files (see Section 5.2.6) are opened and the time and date written on selected output files.

The other four phases are more significant and are described in the following Sections.

PROGRAM NERFFunction

PROGRAM NERF controls the general sequence of operations of the NERF computer program.

Operation

PROGRAM NERF controls the sequence of operations listed in Table 5.1. These operations fall into five phases as identified in Section 5.2.

Generally the role of PROGRAM NERF is to transfer control between the various subroutines that perform the major operations in each phase. The exception is the code associated with the first phase which initialises various output files and initialises the timing routines which accumulate the run time for the computer program.

Note that prior to calling CFIN, PROGRAM NERF reads the first two records of the control file to determine the run identification and to establish whether the run is interactive or batch. This information is required by the code in PROGRAM NERF prior to the call to CFIN which actually re-reads these variables. This seemingly redundant action was unavoidable if CFIN was to be accessible by other programs (such as NERPRE and NERPLT) which may have no reason to access the control file directly.

Phase	Operation	Subroutine
1. Initialisation	Start run-time clock. Open Control file. Open output files (see Section 5.2.6). Write date and time on selected output files.	TIMES
2. Data input and algorithm initialisation	Read data from Control file. Initialise functions and probability densities. Write abbreviated heading on secondary output and data files. Initialise loss factor interpolation. Initialise $p_{ff}(t, n_0)$ nodal arrays. Write heading on primary output file.	CFIN CFIN SETTAB RFSET HEAD
3. Construction of time sequence		ADVNC
4. Calculation of strength distributions		FLPROB
5. Termination	Update Control file with new run number and restart information. Terminate program and output run time.	CFNEW FINISH

Table 5.1. The phases of operation as controlled by PROGRAM
NERF.

5.2.1. Data input and initialisation of computational algorithms

Input data for the NERF computer program is assumed to exist on the files listed in Table 5.2. The control file contains the data and control parameters necessary to define a given run. The function files define the input functions by specifying ordered pairs of function and argument values.

The control file is read by the subroutine CFIN and its format is specified in the documentation for that subroutine. The function files are read by the subroutine READFN which is called during initial entry into each of the functions used to evaluate the input functions. The format for these data files is specified in the documentation for READFN.

The control and function files can be produced directly by using the usual text editing facilities provided by the computer system on which NERF is installed and adhering to the specified formats. However the recommended method is to use the preprocessor NERPRE which provides data editing facilities and by virtue of the fact that it uses CFIN and READFN itself, ensures that the data files are compatible with NERF.

The final input file in the Table, GCOM.DAT, contains a dump of the loss factor interpolation table. This is a binary file and is produced by NERF via subroutine SETTAB. It is not possible for the user to produce this file in any other way.

Setting aside the initialisation of output files and run timing procedures carried out by the code in PROGRAM NERF, the data initialisation phase consists of the following steps.

- (i) The data defining the run is read from the control file and the input functions and probability density functions initialised. This step is executed by CFIN.
- (ii) Default limits are set and, if required, the loss factor interpolation table is constructed. SETTAB performs these functions.

File Name*	Logical Unit No.	Type	Contents	Documentation
CONTRL.IN	7	Control	Control data defining run	Table 5.3.
(CNAME).FCN	2	Function	Ordered pairs for $a(\bar{t})$	Section 5.3.1.
(PNAME).FCN	2	Function	Ordered pairs for $\bar{R}(a)$	Section 5.3.2.
(RNAME).FCN	2	Function	Ordered pairs for $\bar{P}_L(\bar{R})$	Section 5.3.3.
(DNAME).FCN	2	Function	Ordered pairs for $C_d(a)$	Section 5.3.4.
GCOM.DAT	7	Loss factor table	Nodes used for loss factor interpolation	Section 5.4.

Table 5.2. Input data files accessed by NERF.

* A name in parentheses indicates the variable in which the file name is stored in NERF. The name consisting of up to 5 characters is read by NERF from the control file.

245

(iii) RFSET is called to calculate nodes for the evaluation of $P_{ffl}(t, n_0)$.

The operations performed by these routines are described below.

SUBROUTINE CFIN(ICODE)

Function

CFIN reads the control file and initialises the input functions and probability density functions.

Parameter List

ICODE: Control parameter which determines the extent of the operations performed by CFIN, according to the following definitions.

1; The control file is read up to, but excluding, the inspection information.

2; The complete control file is read, but no further processing is made.

3; The complete control file is read and the input functions and probability density functions initialised.

Operation

NERF reads a 'control file' called CONTRL.IN which contains all the control parameters for a given run. The file also contains information necessary to re-commence calculations from a previous run which had terminated prematurely because of run time limitations.

The format of this file is specified in Table 5.3. and the operation of the subroutine in so far as the input of data is concerned, is a direct implementation of this specification.

Data initialisation within CFIN is concerned primarily with the input functions and probability density function. The former are initialised by using the appropriate function routines to activate the initialisation coding within them. For example,

Record	Format	Name	Type	Meaning
1	A5	TITLE(1)	text	Run I.D. (incl. number)
2	L1,G	BATCH	logical	Batch switch
2	G	RTIME	real	Run time (minutes)
3	G	RMUO	real	\bar{R}_0 , median initial strength
4	G	RATEL	real	l_r , load application rate
5	G	CNO	real	a_0 , initial crack length
6	G	KALP	integer	p.d.f. identifier - \underline{X}_3 .
7	G	ALPC1	real	p.d.f. parameters - \underline{X}_1 .
8	G	ALPC2	real	
9	G	ALPC3	real	
10	G	ALPMIN	real	min value for α .
11	G	ALPMAX	real	max value for α
12	G	KBET	integer	p.d.f. identifier - \underline{X}_1
13	G	BETC1	real	p.d.f. parameters - \underline{X}_1
14	G	BETC2	real	
15	G	BETC3	real	
16	G	BETMIN	real	min value for \underline{X}_1
17	G	BETMAX	real	max value for \underline{X}_1
18	G	KCRK	integer	p.d.f. identifier - a_0 (or \underline{X}_2)
19	G	CRK1	real	p.d.f. parameters - a_0 (or \underline{X}_2)
20	G	CRK2	real	
21	G	CRK3	real	
22	A5	RNAME	text	5 chr. name for load function file
23	A5	PNAME	text	5 chr. name for strength function file.
24	A5	CNAME	text	5 chr. name for crack growth function file.
25	A5	DNAME	text	5 chr. name for crack length inspection function file.
26	G	EPS	real	integration convergence criterion
27	G	RLEV	real	Max. no. of levels
28	G	RVALS	real	Max. no. of evaluations
29	8L1	POPLOS	logical	Population losses switch
		FULSV	logical	Full survivorship switch
		CONTI	logical	n.a. (no longer used)
		PERI	logical	Periodic inspection switch

Table 5.3. Specification for the control file, CONTRL.IN.

Record	Format	Name	Type	Meaning
29 (cont.)	8L1	LIMRSK	logical	Limit risk inspection switch
		INSLOS	logical	replacement switch
		VIRGIN	logical	uncracked risk switch
		RESTART	logical	re-start switch
30	3L1	ALPCON	logical	constant α switch
		BETCON	logical	constant \underline{X}_1 (or β) switch
		RNOCON	logical	constant a_0 (or n_0) switch
31	G	NMAX	integer	N, number of evaluation times
32 to 32+NMAX	.G	RNSVAL		Array of evaluation times, t_n
33+NMAX	G	NDMAX	integer	M, number of strength distribution times.
34+NMAX to 34+NMAX+NMAX	G	RNDIST	real	t_{Rm} , strength distribution times
End of first group of data				
35+NMAX+NMAX	G	NIMAX	integer	J, number of inspection times
36+NMAX+NMAX to 36+NMAX+NMAX+NIMAX =36+NTOT	G	RNINS	real	t_{ij} , inspection times
37+NTOT	G	CND	real	a_d crack length inspection criterion
38+NTOT	G	PLD	real	R_p , proof load inspection criterion
39+NTOT	G	RLMRSK(1)	real	first limit risk
40+NTOT		RLMRSK(2)	real	second limit risk
41+NTOT to 56+NTOT	G	RSK	real	contents of common block, RISKCM; re-start information
End of second group of data				

Table 5.3. Continued.

XXX = CRKGR(0.0)

will initialise the crack growth function, (and assign XXX the value $a_0(0)$). The latter are initialised by calling the appropriate setting routines, (e.g. BETSET to initialise $p_s(\beta)$) followed by the subroutine RANGE to find the sensible limits for the random variable. These limits override those input via the control file only if they represent a tighter restriction of the random variable. In the case of initial crack length, the input limits are assumed to be such that initial age lies in the interval $[\bar{\tau}_1, \bar{\tau}_f]$, (corresponding to the default limits (3.48) and (3.49)), and following initialisation, a normalisation factor is computed using the adaptive integration function ADAPT2. The density function is then modified via the function RNONRM. For further details of the initialisation of the density functions refer to the documentation of relevant functions in Section 5.4.4.

Several minor data initialisation operations are executed by CFIN. These are itemised below.

- (i) The integration control parameters are transferred to the relevant storage locations in each of the three adaptive integration routines which are also initialised via the subroutine ADASET.
- (ii) If population losses are ignored by the model (POPLOS = true), then $P_s(t)$ must be computed from $r(t)$. In this case, FULSV is set false, overriding any setting given by the user.
- (iii) The re-start option is not permitted if RNSOLD, (the last value of t) is zero. In this case RESTRT is set false, overriding the setting in the control file.
- (iv) The uncracked structures switch, VIRGIN, is set true only when $\bar{\tau}_1 > 0$, $n_0 = 0$ and VIRGIN is true in the control file.

- (v) A model is flagged as having constant initial crack length if either $CN0$ (a_0) is set to zero in the control file or the switch $RNOCON$ has been set true.
- (vi) $NOUT$ is set to identify the outermost level of integration.

SUBROUTINE RFSET(RFARG,IK)

Function

The evaluation of $p_{ff1}(t, n_0)$ involves an integration along the line $R_{min} = \alpha \psi(\beta)$. This integration has been found to require that nodal values for α which correspond to those in β used to define $\psi(\beta)$ are used as a basis for the adaptive integration strategy. Subroutine RFSET calculates those nodes.

Parameter List

RFARG: Array into which the α nodes are returned to the calling code.

IK: Number of nodes in RFARG.

Operation

The limits of integration for α are (from equation (3.97)), $R_{min}/\psi(\beta_{p, R_{min}})$ and $\alpha_2(\tilde{t}_f^*(R_{min}), R_{min})$. Ignoring the effects of the proof load inspection boundary and variations in n_0 , the limits for the α nodes are,

$$\max\{\alpha_{min}, R_{min}/\psi(\tilde{t}_i)\} \leq \alpha \leq \min\{\alpha_{max}, R_{min}/\psi(\tilde{t}_f)\} \quad (5.1)$$

Because of the limiting assumptions made by the function PSINV, corresponding β limits can be found by,

$$\begin{aligned} \beta_{min} &= \text{PSINV}(R_{min}/\alpha_{min}) \\ &= \max\{\tilde{t}_i, \psi^{-1}(R_{min}/\alpha_{min})\} \end{aligned} \quad (5.2)$$

and

$$\begin{aligned} \beta_{max} &= \text{PSINV}(R_{min}/\alpha_{max}) \\ &= \min\{\tilde{t}_f, \psi^{-1}(R_{min}/\alpha_{max})\} \end{aligned} \quad (5.3)$$

These limits are calculated by RFSET and used, via the

subroutine MERGE, to produce a vignettted sequence of β nodes. The β nodes are then converted to α nodes, (using the relationship $\alpha = R_{\min} / \psi(\beta)$).

Note that RFSET must be called after SETTAB to ensure that the correct default limits have been set.

SUBROUTINE SETTAB

Function

Subroutine SETTAB sets the default limits for R and β and initialises the loss factor interpolation table. The subroutine also controls the construction of contour maps of the loss factor functions.

Operation

A flow diagram of the major operations performed by SETTAB is shown in Figure 5.1. Those operations associated with the initialisation of the loss factor interpolation table are described in Section 5.4.2. The description presented here pertains specifically to the calculation of the default limits, the graphical operations and the more mundane aspects of the subroutine documentation (e.g. COMMON variables changed).

(1) Default limits

The default limits are defined by equations (3.43) to (3.49). Those for n_0 are set by CFIN, or following an inspection, ADVNCE. The remainder are set during the first section of the code in SETTAB.

Using primes to denote the input values of variables, the limits for R are

$$\begin{aligned} R_{\max} &= RMAX = \min(\alpha_{\max}, R'_{\max}) \\ &= AMIN1(ALPMAX, R(M)) \end{aligned} \quad (5.4)$$

and

$$\begin{aligned} R_{\min} &= RMIN = \max(\alpha_{\min} * \psi(\bar{E}_f), R'_{\min}) \\ &= AMAX1(ALPMIN * PSI(RNF), R(1)) \end{aligned} \quad (5.5)$$

After setting these limits, the interpolation tables

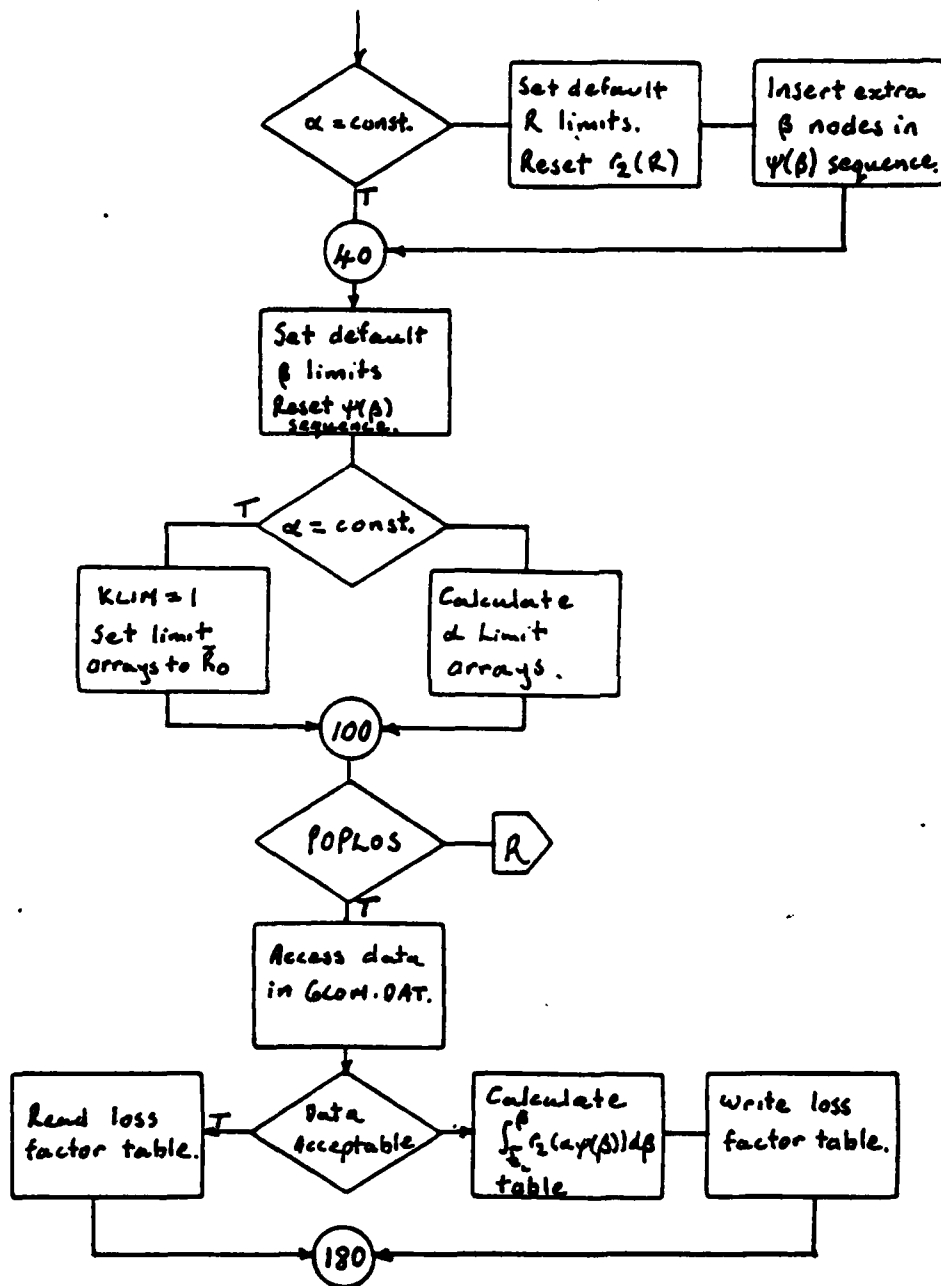


Figure 5.1. Flow diagram for the major operations performed by SETTAB.

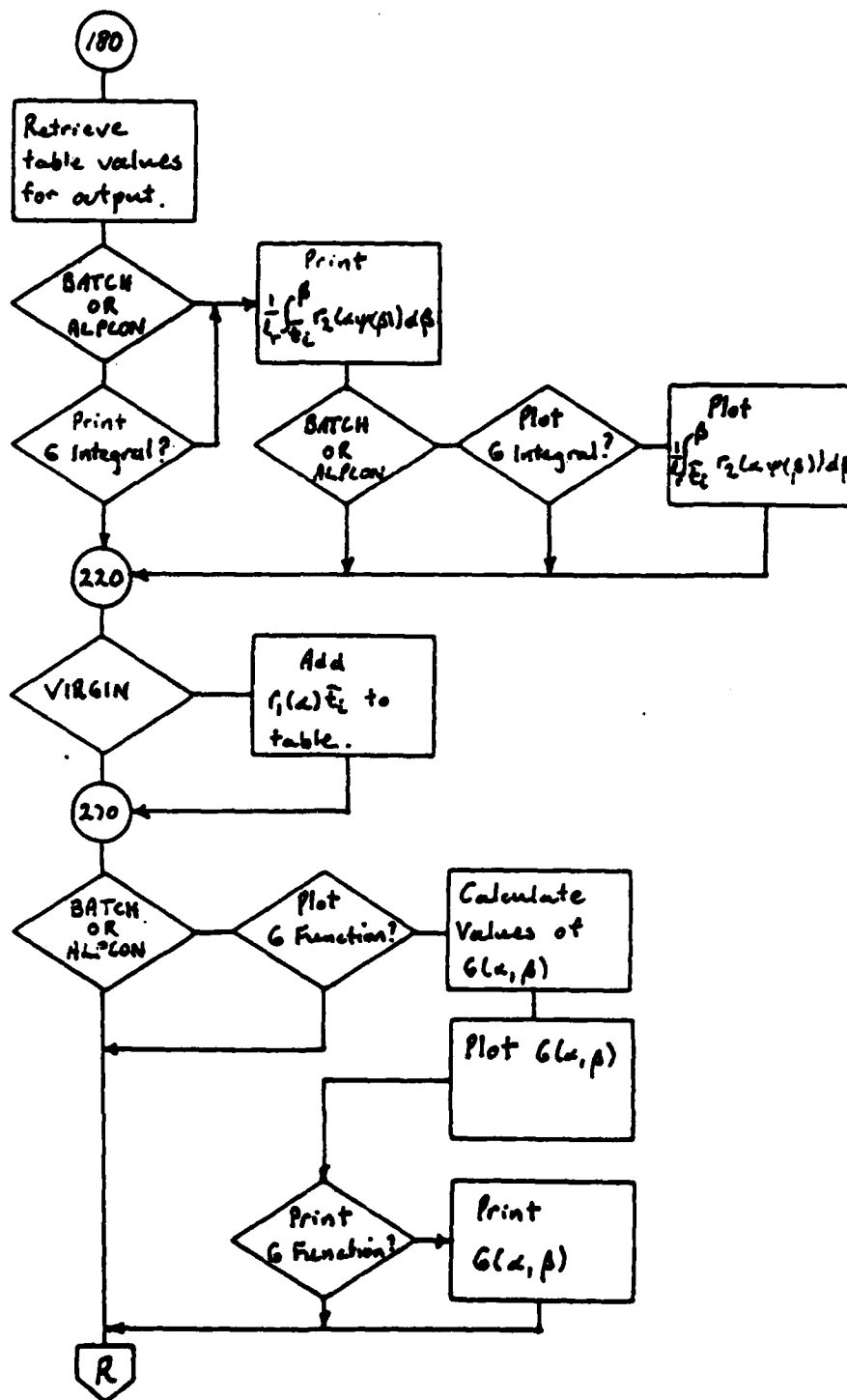


Figure 5.1. Continued.

for $r_2(R)$ are altered accordingly using the subroutine MERGE to vignette the sequence of ordered pairs and function RLOSET to recalculate the interpolation tables.

The default limits for β are,

$$\begin{aligned}\tilde{t}_1 = RNI &= \max\{\tilde{t}_1', \psi^{-1}(R_{\max}/\alpha_{\min})\} \\ &= \text{AMAX1}(RNI, \text{PSINV}(R(M)/\text{ALPMIN}))\end{aligned}\quad (5.6)$$

and

$$\begin{aligned}\tilde{t}_f = RNF &= \min\{\tilde{t}_f', \psi^{-1}(R_{\min}/\alpha_{\max})\} \\ &= \text{AMIN1}(RNF, \text{PSINV}(R(M)/\text{ALPMAX})).\end{aligned}\quad (5.7)$$

Subroutine MERGE is used to vignette the ordered pairs defining $\psi(\beta)$ and to include three additional pairs having β values, $\psi^{-1}(R_{\max}/\alpha_{\max})$, $\psi^{-1}(R_{\min}/\alpha_{\min})$ and $\psi^{-1}(R_{\min}/R_{\max})$ which are used in the construction of the loss factor interpolation table (Section 5.4.1.) The interpolation tables for $\psi(\beta)$ are then modified using PSISSET (Section 5.3.2).

(ii) Loss factor table

When the loss factor interpolation table is constructed, SETTAB dumps a binary record of it in a file called GCOM.DAT. Before constructing any table SETTAB accesses this file and checks whether the existing table is satisfactory. If the variables listed in table 5.4. are in agreement with those recorded in GCOM.DAT, the construction of a new table is bypassed.

(iii) Contour maps of loss factor functions

SETTAB provides for the optional printing and contour map construction of the functions

Variable Name	Significance
RNAME	Name of file containing $\bar{P}_L(R)$
PNAME	Name of file containing $R(a)$
CNAME	Name of file containing $a(\bar{t})$
RNI	\bar{t}_1 (after default limiting)
ALPMIN	α_{\min} Lower limit for α
ALPMAX	α_{\max} Upper limit for α
EPSINT	ϵ_{rel} Error criterion for adaptive integration
N	Number of nodes defining $\psi(\beta)$, after range adjustment by SETTAB
KLIM	Number of nodes allocated for α direction interpolation for the loss factor table, see Section 5.4.

Table 5.4. Variables tested for acceptance of a loss factor table stored in GCOM.DAT.

$$\int_{\tilde{\epsilon}_c}^{\beta} \frac{r_2(\alpha\psi(\beta))d\beta}{L_r} \quad \text{and} \quad \frac{1}{\beta} \left[r_1 \tilde{\epsilon}_c + \int_{\tilde{\epsilon}_c}^{\beta} r_2(\alpha\psi(\beta))d\beta \right] ,$$

called the 'G integral' and the 'G function' respectively. The significance of these functions is described in Section 5.4. The last section of code in SETTAB controls the construction of the maps using the subroutine INTPLT.

COMMON Variables Changed

BETA: Array of β values defining $\psi(\beta)$.

PS: Array of ψ values defining $\psi(\beta)$.

N: Number of ordered pairs defining $\psi(\beta)$.

R: Array of R values defining $r_2(R)$.

RLD: Array of values of $\log(r_2(R))$.

M: Number of ordered pairs defining $r_2(R)$.

RN0: Current value of n_0 .

G: Two-dimensional array containing loss factor interpolation table.

KLIM: Number of rows in the loss factor interpolation table.

ALP1: Array of lower α limits for loss factor interpolation.

ALP2: Array of lower α limits for loss factor interpolation.

BETV: Current value of β .

ALPV: Current value of α .

GEXP: Work space in Block GCOM.

Significant Local Variables

RMIN: Minimum value of R.
 RMAX: Maximum value of R.
 BETLOW: Minimum value of β .
 BETHIG: Maximum value of β .
 PSIV: Current value of $\psi(\beta)$.
 ICOUNT: Integer array containing the numbers of function evaluations used to calculate the values in the loss factor interpolation table. (Shares space with GEXP.)
 GEXP: Used during the last stages of SETTAB to store the loss factor functions prior to printing or contour map construction.

Prompts

- (i) 'R LIMITS BEING CHANGED': The default limit equations imply a change to the range of R.
- (ii) 'TI IS BEING CHANGED': The default limit equations imply a change to $\tilde{\tau}_1$.
- (iii) 'TF IS BEING CHANGED': The default limit equations imply a change to $\tilde{\tau}_f$.
- (iv) 'DATA FILE NOT ACCEPTABLE ... G BEING CALCULATED': The data in GCOM.DAT is not acceptable.
- (v) 'G ARRAY READ FROM DISK FILE': The data in GCOM.DAT is acceptable.
- (vi) 'PRINT G INTEGRAL?': Request for user response whether to print the G integral.
- (vii) 'PLOT G INTEGRAL?': Request for user response whether to construct contour maps of the G integral.

- (viii) 'PRINT G FUNCTION?': Request for user response whether to print the G function.
- (ix) 'PLOT G FUNCTION?': Request for user response whether to construct a contour map of the G function.

5.2.2. Development of a time sequence including inspections

Following the data initialisation phase, the time sequence of reliability functions can be constructed. This phase, which accounts for most of the computational effort made by NERF is controlled by the subroutine ADVNCE.

The reliability functions are computed at time values $\{t_k\}$ which are a combination of the sequences $\{t_n\}$, (evaluation times), $\{t_{i_j}\}$, (inspection times) and $\{t_{R_m}\}$, (strength distribution times) as outlined in Section 3.6. These sequences are specified as part of control file data or, in the case of $\{t_{i_j}\}$ and $\{t_{R_m}\}$, can be computed by NERF.

During this phase, all the reliability functions save for the strength distributions are computed at each value of t_k . Sufficient data is stored at each value of t_{R_m} to permit the code associated with the following phase to compute the strength distributions with a minimum of effort.

Although the overall construction of the time sequence is controlled by the subroutine ADVNCE, function RSKTOT which computes the set of reliability functions at a given value of time makes a significant contribution to the generation of the time sequence.. This function is therefore included in the descriptions presented below.

SUBROUTINE ADVNCE

Function

Subroutine ADVNCE controls the construction of a time sequence of reliability functions.

Operation

ADVANCE contains two main sections of code. The first initialises all the variables required during the development of the time sequence. The second section constructs the sequence.

(1) Section 1

The operations performed by the first section of code in ADVANCE are best described in the order they appear in the code. The paragraphs below correspond approximately to the various sub-sections of the code, (as identified by comments in the program listing).

The set of reliability functions and various related parameters are stored in the COMMON block RISKCM. Although the variables in all COMMON blocks are defined in Appendix A.2., those in RISKCM are listed in Table 5.5. for easy reference.

The first set of operations are associated with initialisation of flags and preliminary output. The flags INTOFF and the switch OUTINT are given default values and the subroutine OUTPUT is called to provide appropriate headings on the primary output file.

The merging of the two sequences $\{t_n\}$ and $\{t_{Rm}\}$ commences with the setting of the three switches FLSTOR, BEFORE and AFTER which are used by subsequent coding to store data at specified values of t_{Rm} , store data just before each inspection or store data just

Variable Name	Symbol	Significance
RLT	$P_{\underline{S}}(t)$	Probability of Survival
RSLT	$r_s(t) \cdot P_{\underline{S}}(t)$	
RFLT	$r_f(t) \cdot P_{\underline{S}}(t)$	
RTOT	$r(t)$	Total risk rate
RS	$r_s(t)$	Risk of static fracture
RF	$r_f(t)$	Risk of fatigue life exhaustion
RV	$r_v(t)$	Virgin risk
AVGRS	$E(\underline{F}) \cdot P_{\underline{F}}(t)$	
AVGLT	$E(\underline{F})$	Expected time to failure (Equation 3.176).
RLTINS	$\sum_{j=1}^{j'} P_{\text{det}}(ti_j)$	Fraction of population removed by all previous inspections
RNSOLD	t	Within RSKTOT these have the value at the list time of evaluation
ROLDLT	$p_{\underline{F}}(t)$	
RNIM	ti_j	time of last inspection
RDET	$P_{\text{det}}(ti_j)$	Probability of detection at the last inspection (Calculated by ADVNCE)
RMEAN	$r_{\text{mean}}(t)$	Mean risk (equation 3.175)

Table 5.5. Variables stored in the common block RISKCM.

Unless otherwise specified, definitions refer to value on exit from RSKTOT.

after each inspection respectively. If the either of the last two switches are true, the sequence $\{t_{Rm}\}$ is empty and no merging is necessary. Otherwise the necessary merge is controlled by the subroutine MERGE.

It is possible to 're-start' NERF following premature termination of a previous run. If the switch RESTART is true, INDLOW is used to find the first value in the evaluation time sequence which is greater than the last value computed (RNSOLD). Otherwise RSTART, (the first time value) is set to RNSVAL(1).

Similarly the first specified inspection time following the last time of evaluation must be found and the relevant parameters set accordingly. If a restart is not required, those parameters are set to their initial values. The relevant parameters, their variable names and initial values are listed in Table 5.6. Note that if the limit risk and periodic inspection switches are both set true, then only one time value is allowed in the specified inspection time sequence $\{t_i\}$ (RNINS).

If a restart is specified, then the starting value of the reliability functions are assumed to be stored in RISKCM, having been read from the control file. Otherwise, the initial values corresponding to time $t=0$ are set. Because the values for $t=0$ are not considered to be part of the program output, initial values are written on the primary output file (via OUTPUT) in the case of a restart only.

The final part of the code in the first section of ADVNCE writes an appropriate warning on the output files if $P_S(t)$ is being computed from $r(t)$ by approximate integration, checks that at least one time value has been specified and in the case of an interactive run for which a single time value is

Variable Name	Symbol	Value
RLT	$P_S(t)$	1
RTOT	$r(t)$	0
RS	$r_S(t)$	0
RV	$r_V(t)$	0
RF	$r_f(t)$	0
RMEAN	$r_{\text{mean}}(t)$	0
AVGRS	$E(\underline{F})P_{\underline{F}}(t)$	0
AVGLT	$E(\underline{F})$	0
RLTINS	$P_{\text{det}}(ti_j)$	0
RNS	t	0
RNSOLD	t_{k-1}	0
ROLDLT	$p_{\underline{F}}(t)$	SMALL (10^{-15})

Table 5.6. Variables initialised by ADVNCE and their initial values.

specified, seeks a response from the user to activate the optional graphics support functions provided by NERF.

(11) Section 2

The second section of code in ADVNCE evaluates the time sequence of reliability functions. Given the already merged sequences, $\{t_n\}$ and $\{t_{Rm}\}$ stored in the array RNSVAL, the bulk of the code in this section is concerned with the inclusion of the inspection times and accounting correctly for the effects of the inspections according to the options that have been selected.

Most of the code in this section is contained within a DO LOOP which terminates at statement labelled 290 and sweeps through the sequence of time values stored in RNSVAL. The logic associated with the embedding of the inspection times within the overall sequence is shown in Figure 5.2.

As each value of t_k is selected from RNSVAL, it is checked, initially to ensure that it is not zero, and then to determine if an inspection precedes it. In the case of periodic inspections, this is simply a matter of accessing the time of the next inspection, stored in RINSP.

If the limit risk procedure has been selected, the total risk at the current value of t_k is computed and compared with the limit risk (RTEST). If the risk is too large, an inspection is inserted in the interval (t_{last}, t_k) , where t_{last} is the last time value at which the reliability functions were evaluated, (which is not necessarily t_{k-1}). This inspection time is found by solving the equation

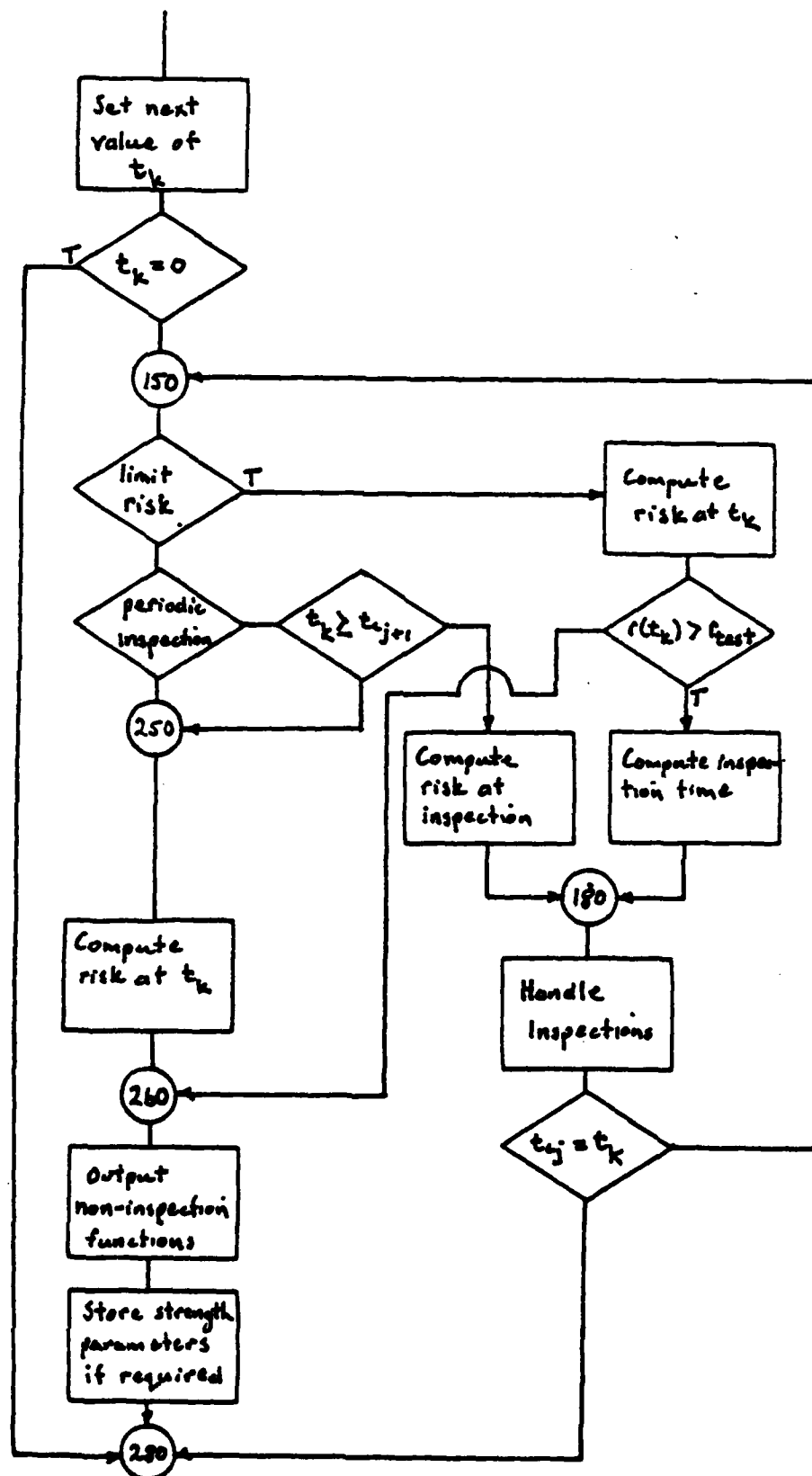


Figure 5.2. Logic associated with the embedding of inspection times in the overall sequence.

$$\log(r(t)) = \log(RTEST) \quad (5.8)$$

using the function FSOLVE as described in Section 4.3.

Once the inspection time has been computed (by either method) the inspection is incorporated into the model. This operation is expanded in Figure 5.3 and is detailed more precisely in (iii) below. Control is then returned to the point where t_k has just been selected and the whole process is repeated until no further inspections can be included before t_k .

Note that if following a limit risk inspection the risk rate cannot be reduced below RTEST, the limit risk process is terminated and no further inspections are attempted.

Once all the inspections preceding t_k have been found, the reliability functions are computed for $t=t_k$. If FLSTOR is true and t_k corresponds to a value in RNDIST, $p_F(t)$, $P_S(t)$ and ti_j are stored in the next available locations in RSLTDS, RLTDs and RNIMDS respectively.

The code, excluding that between statements 180 and 250 which are described below, follows the logic in Figure 5.2 in an obvious manner. The code relies on the function RSKTOT to return $r(t)$ at the current value of time and to update the reliability functions stored in the COMMON block RISKCM. The subroutine OUTPUT is used to record these functions on the primary output file. (Section 5.2.6.)

(iii) Inspections

Once an inspection time has been determined, the effect of the inspection is incorporated into the model by the logic shown in Figure 5.3. This logic controls the output of the reliability functions just before the

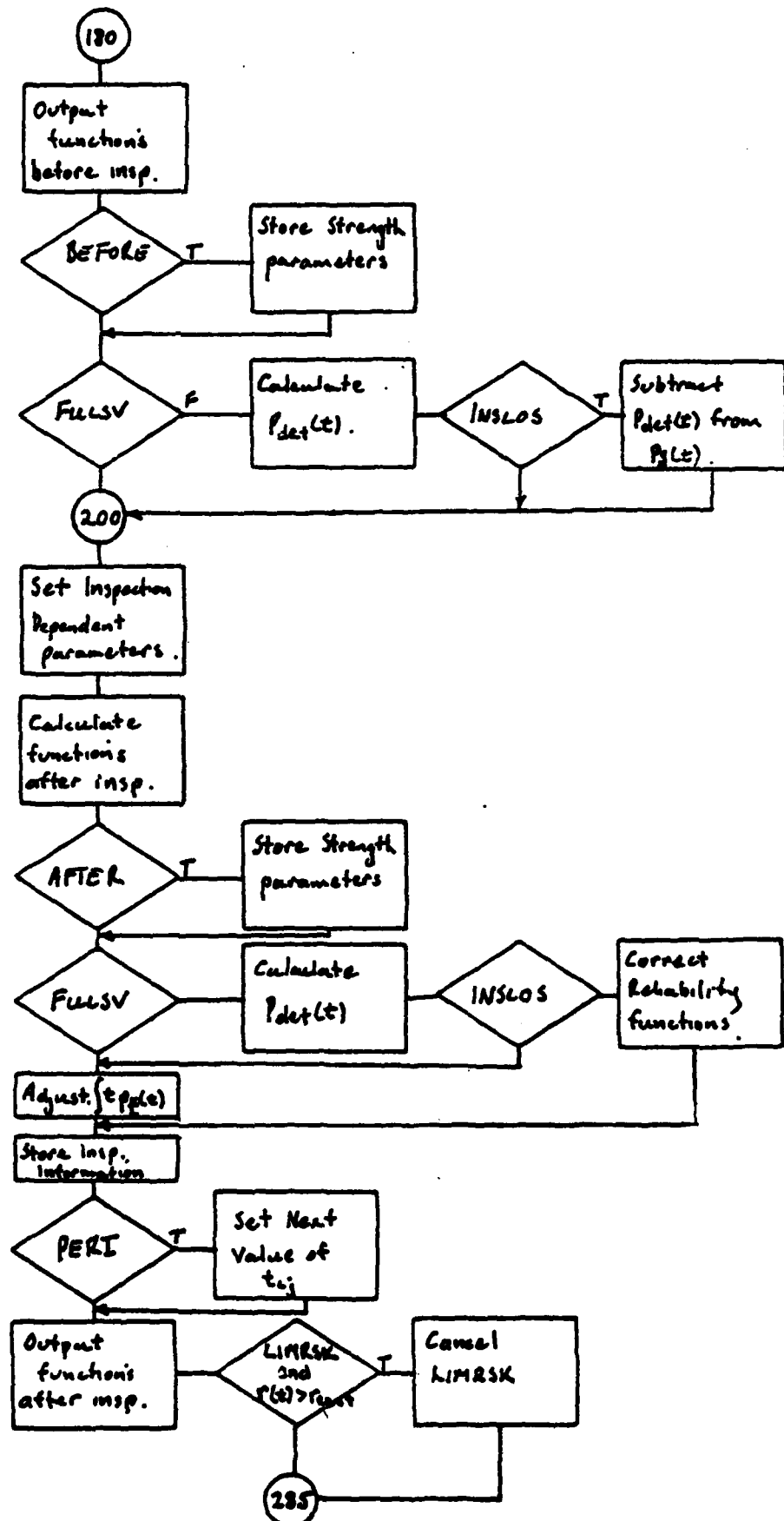


Figure 5.3. Logic associated with incorporating the effect of an inspection in the model.

inspection, the adjustment of the inspection dependent parameters in the model and the calculation of the reliability functions immediately following the inspection. Provision is also made for the storage of the parameters required to compute the strength distributions either before or after each inspection.

This code also calculates the probability of detection, $P_{det}(t)$, at the inspection. If $P_S(t)$ is calculated by an approximate integration of $\bar{r}(t)$, (FULSV=true), then an integral expression for $P_{det}(t)$ must be used. This is controlled by the function FDET which must be called prior to the alteration of the inspection dependent parameters of the model. If there is no replacement at the inspection, $P_{det}(t)$ is subtracted from $P_S(t)$, (RLT). Note that once this change to RLT has been made the effect is permanent only when $P_S(t)$ is calculated by integrating $r(t)$.

If $P_S(t)$ is calculated via a full integral expression for $P_F(t)$, then $P_{det}(t)$ can be evaluated by taking the difference between $P_S(t)$ just before and just after the inspection. This calculation must, of course, be made after the inspection dependent parameters have been changed.

If structures removed during an inspection are replaced by 'perfect structures' (see Section 3.6.2.) the probability of survival is corrected by equation (3.183) in function RSKTOT. The sum of the $P_{det}(t)$ terms for all previous inspections is stored in RLTINS which is updated by ADVNCE. This means that the value of $P_S(t)$ calculated by RSKTOT just after the inspection is in error and appropriate adjustments to all functions depending on $P_S(t)$ for their evaluation are made.

Graphics Operations

During an interactive run which involves a single evaluation time, NERF provides the facility for constructing various graphical representations of the integrations. ADVNCE seeks a response from the user regarding this option.

Significant Local Variables

RNDS: Stored value of \tilde{t}_d .

RND1: Maximum value of t_d as determined by the proof load limit, $\psi^{-1}(R_p/\alpha_{\max})$.

FLSTOR: Logical switch. If true strength distribution data is stored at the specified times stored in RNDIST.

AFTER: Logical switch. If true, strength distribution data is stored just after each inspection.

BEFORE: Logical switch. If true, strength distribution data is stored just before each inspection.

M1: Temporary upper limit for NMAX.

RNSMIN: Minimum value for $\{t_n\}$.

RNSMAX: Maximum value for $\{t_n\}$.

IDCURR: Location of the next free storage areas for strength distribution data.

RSTART: Starting value of time for the current run.

ISTART: Initial value of k , (index for $\{t_k\}$) for the current run.

RINSP: Time of the next inspection.

RTEST: Current value of limit risk.

RNIM: Last inspection time.

R1: Logarithm of the last value of total risk, $r(t)$.

I: Main DO LOOP index, = k.

Prompts

- (1) ' WARNING FOR SURVIVORSHIP CALCULATION
IF THE TIME INTERVALS ARE LARGE
FULL SURVIVORSHIP SHOULD BE USED':
Written only when FULSV is false. Warns use that
integration of $r(t)$ to yield $P_S(t)$ is approximate.
- (ii) 'DURING THE CALCULATION OF RISK
ARE PLOTS REQUIRED'
Prompts the user to select the optional information
plotting facilities. Appears only when a single
evaluation time has been specified and during an
interactive run.
- (iii) 'INTEGRATION INFORMATION PLOTTED' Echoes the user's
request to plot integration information.
- (iv) 'FINDING RNS CORRESPONDING TO LIMIT RISK': Notification
that FSOLVE is controlling the time that the
reliability functions are being evaluated to find a
limit risk inspection.
- (v) 'RISK AT INSPECTION': Notification that any following
prompts are associated with the evaluation of the
reliability functions just prior to an inspection.
- (vi) 'RISK FOLLOWING AN INSPECTION': Notification that any
following prompts are associated with the evaluation of
the reliability functions just after an inspection.

Error Messages

- (1) 'NO TIME VALUES': There are no evaluation or strength
distribution times specified. Processing is
terminated.

- (11) 'RISK AFTER INSPECTION GREATER THAN LIMIT RISK':
Inspection has failed to reduce the risk rate below
limit. No further inspections are attempted.

FUNCTION RSKTOT(RNS)

Function

RSKTOT returns the value of $r(t)$ for a given value of time, RNS. The function also evaluates the reliability functions $P_S(t)$, $r_V(t)$, $r_g(t)$, $r_f(t)$, $r_{mean}(t)$ and $E(F)$ together with other time dependent functions used in the derivation of the reliability functions and stores the results in the COMMON block RISKCM (see Table 5.5).

Parameter List

RNS: Value of time for which the evaluation of $r(t)$ is required.

Operation

The code in RSKTOT follows the logic shown in figure 5.4 and commences operations by writing a heading for the current value of time via the subroutine PROMPT and constructing, if required, a contour map of the loss factor. The remainder of the operations is simply a systematic progression through the evaluation of the required reliability functions.

Having set the COMMON variable RNSV to the new value of time and transmitting that value to the functions which evaluate $p_g(\beta)$ (via BETCNG), RSKTOT evaluates the three functions $r_g(t)P_S(t)$, $r_f(t)P_S(t)$ and $r_V(t)P_S(t)$ which are stored in RSLT, RFLT and RVLT respectively. Depending on the class of model, the first two functions are evaluated either by integrations over initial crack length using ADAPTO or by making point evaluations of the relevant integrand functions. In the former case, integration limits for a_0 and n_0 are found according to the logic described in Section 5.6 below. The third function is evaluation by a single call to FRV0.

If FULSV is true, so that $P_S(t)$ is calculated via an integral expression for $P_F(t)$, ADAPTO is used again, using the

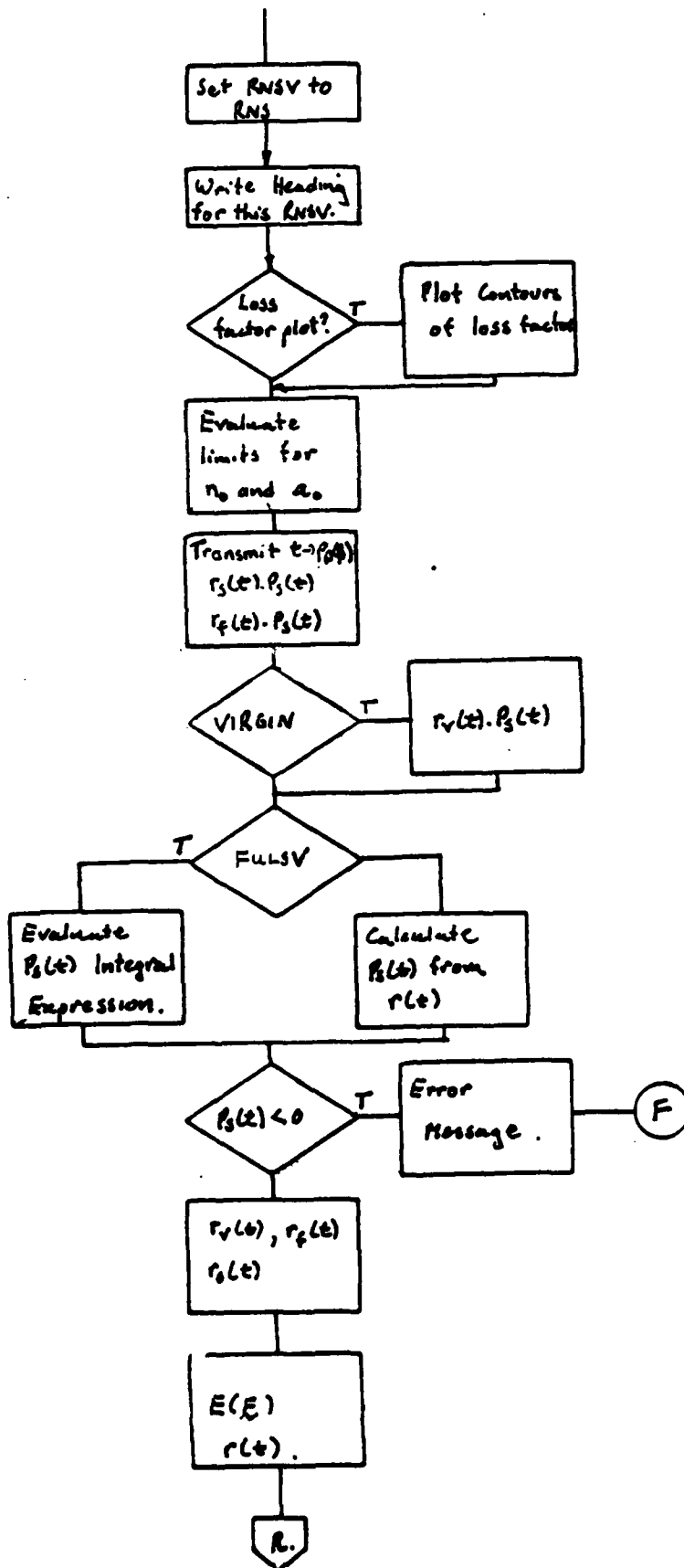


Figure 5.4. Sequence of major operations executed by RSKTOT.

same limits as used for $r_s(t)P_s(t)$. If replacement at inspections has been selected the cumulative probabilities of detection for previous inspections are added to $P_s(t)$.

If FULSV is false, $P_s(t)$ is calculated by integrating the risk rate using equation (3.186). This evaluation makes use of the following equations which define relevant local variables.

$$RLT = RLT_{last} - RLTDEL \quad (5.9)$$

where

$$RLTDEL = \frac{(p_F(t) - p_F(t_{last})) * DNS}{(\log(p_F(t)/p_F(t_{last})))} \quad (5.10)$$

$$DNS = t - t_{last} = RNSV - RNSOLD \quad (5.11)$$

$$p_F(t) = RTOTLT \quad (5.12)$$

and

$$p_F(t_{last}) = ROLDLT. \quad (5.13)$$

Note that it is not necessary that $t > t_{last}$. The integration can yield acceptable results during the determination of a limit risk inspection time. However, it is recommended that the more accurate full integration for $P_s(t)$ is used for limit risk calculations.

If RLT drops below zero, the operation of NERF is terminated.

The calculation of the expected time to failure follows equation (3.178) with the following,

$$\begin{aligned}
 \text{AVGRS} &= t_{\underline{F}}(t) \\
 &= \text{AVGRS}_{\text{last}} + \text{CON1} * (t \cdot p_{\underline{F}}(t) - t_{\text{last}} \cdot p_{\underline{F}}(t_{\text{last}})) \\
 &\quad + \text{CON1} * (\bar{p}_{\underline{F}}(t) - \bar{p}_{\underline{F}}(t_{\text{last}})) \quad (5.14)
 \end{aligned}$$

$$\begin{aligned}
 \text{CON1} &= (t - t_{\text{last}}) / (\log(p_{\underline{F}}(t) / p_{\underline{F}}(t_{\text{last}}))) \\
 &= (\text{RNS} - \text{RNSOLD}) / \text{ALOG}(\text{PDF} / \text{PDFOLD}). \quad (5.15)
 \end{aligned}$$

The expected time to failure is then given by

$$\begin{aligned}
 E(F) &= \text{AVGLT} \\
 &= \text{AVGRS} / \text{RLT} \quad (5.16)
 \end{aligned}$$

RSKTOT completes its operation by computing the functions $r_v(t)$, $r_f(t)$, $r_s(t)$ and $r(t)$ which can be obtained trivially from the functions already computed.

Prompts

- (i) 'INFORMATION ----- NS = *****
Heading to indicate that a new time value is relevant.
- (ii) 'PLOT LOSS FACTOR?': Prompt to seek user response whether to plot contour map of loss factor.
- (iii) 'RISK FUNCTION RSLT': Notification that any subsequent prompts are associated with the calculation of $r_s(t)P_{\underline{S}}(t)$.
- (iv) 'RF. TERM': Notification that calculation of $r_f(t)$ has commenced.
- (v) 'PROBABILITY OF FAILURE': Notification that the calculation of $P_{\underline{F}}(t)$ has commenced.

AD A145 685

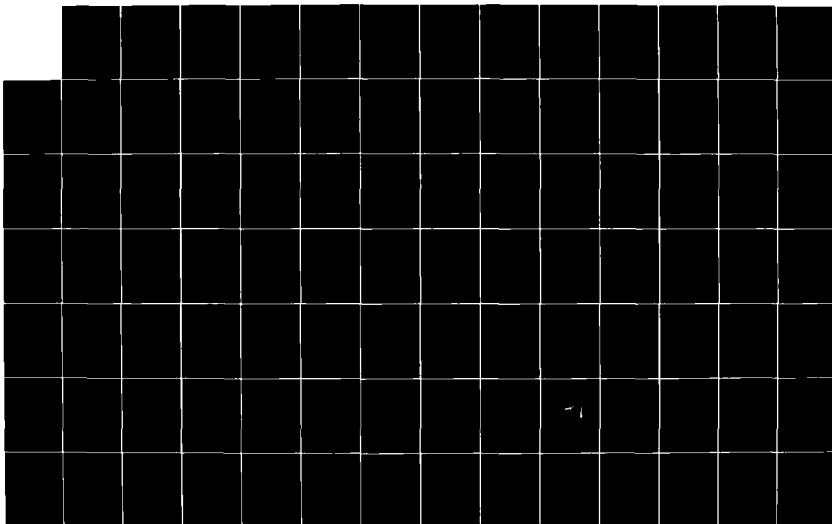
NERI A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUN... (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARI/STRUC 397

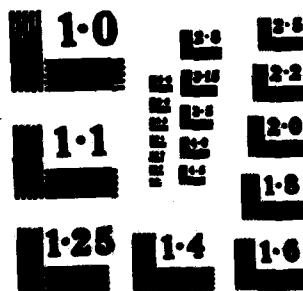
4/7

UNCLASSIFIED

F/G 9/2

NI





278

Error Message

(1) 'PROBABILITY OF SURVIVAL HAS REDUCED TO 0.0': $P_s(t)$ has
reduced to below zero. NERF is terminated.

FUNCTION RSKLOG(ARG)

Function

RSKLOG returns the value of $\log(r(t))$ for a given value of time (ARG)

Parameter List ARG: Value of time for which the evaluation is required.

Operation

RSKLOG relies on the function RSKTOT to calculate $r(t)$ and then computes the logarithm in a trivial manner.

5.2.3. Calculation of strength distributions

The third phase of operation of NERF is the evaluation of reliability functions related to variations in strength. The basic strength functions are the density functions $p_R(R|t)$ and $p_R(R|\underline{F}>t)$. The corresponding distributions $P_R(R|t)$ and $P_R(R|\underline{F}>t)$ and expected values, $E(R|t)$ and $E(R|\underline{F}>t)$ are defined by equations (3.181) to (3.184) and can be calculated from the density functions.

Upon completion of the previous phase, the sequence of time for which the strength functions are required $\{t_{Rm}\}$ is known and the values of $P_S(t)$, $p_F(t)$ and t_{ij} for each of these times has been stored in the COMMON block DISCOM.

The calculation of the strength distributions is controlled by the subroutine FLPROB which essentially constructs tables of values of $p_R(R|t)$ and $p_R(R|\underline{F}>t)$ for the specified values of t_{Rm} and for values of R in the interval $(\max(R_{\min}, \alpha_{\min}(\tilde{t}_f)), \min(R_{\max}, \alpha_{\max}))$. Once these tables have been constructed and output to the primary output file and to function files for subsequent plotting, the table values are integrated to produce the remaining functions which are then similarly output.

The operation of FLPROB is described below.

SUBROUTINE FLPROB

Function

FLPROB controls the calculation of the reliability functions that are related to the variation in strength. These functions are; $p_R(R|t)$, $P_R(R|t)$ and $E(R|t)$ the density, distribution and expected value of the failing load and $p_R(R|F>t)$, $P_R(R|F>t)$ and $E(R|F>t)$, the density, distribution and expected value of strength.

Operation

FLPROB contains code which is directly related to the evaluation of $p_R(R|t)$ and $p_R(R|F>t)$. This code is detailed in Section 5.8. The remainder of the code is concerned with the broader aspects of the construction of tables of the strength functions and is the subject of this description.

The sequence of operations follows the logic shown in Figure 5.5. On initial entry into the subroutine, the limits for R (RMIN,RMAX) are calculated and the storage arrangements for the tables of functions established. Both $p_R(R|F>t)$ and $p_R(R|t)$ are calculated for NDMAX values of time and M1 values of R. NDMAX is the number of values in the sequence $\{t_{Rm}\}$ and M1 is the number of values of R. M1 is set within the code in FLPROB and is currently set at 101. The tables are organised as N1 columns of M1 values where $N1=NDMAX+1$, the first column containing the M1 values of R.

Both tables occupy space in the GEXP array (COMMON block GCOM). $p_R(R|t)$ is stored in the first $M1*N1$ locations, $P_R(R|F>t)$ in locations 1251 to 1250 + $M1*N1$. (This arrangement assumes that there are at most 10 values in $\{t_{Rm}\}$ and that GEXP as been allocated 2500 words..

The two density functions are evaluated simultaneously in a nested DO LOOP sequence. The outer loop sweeps through $\{t_{Rm}\}$; the inner loop through the values of R. As mentioned above,

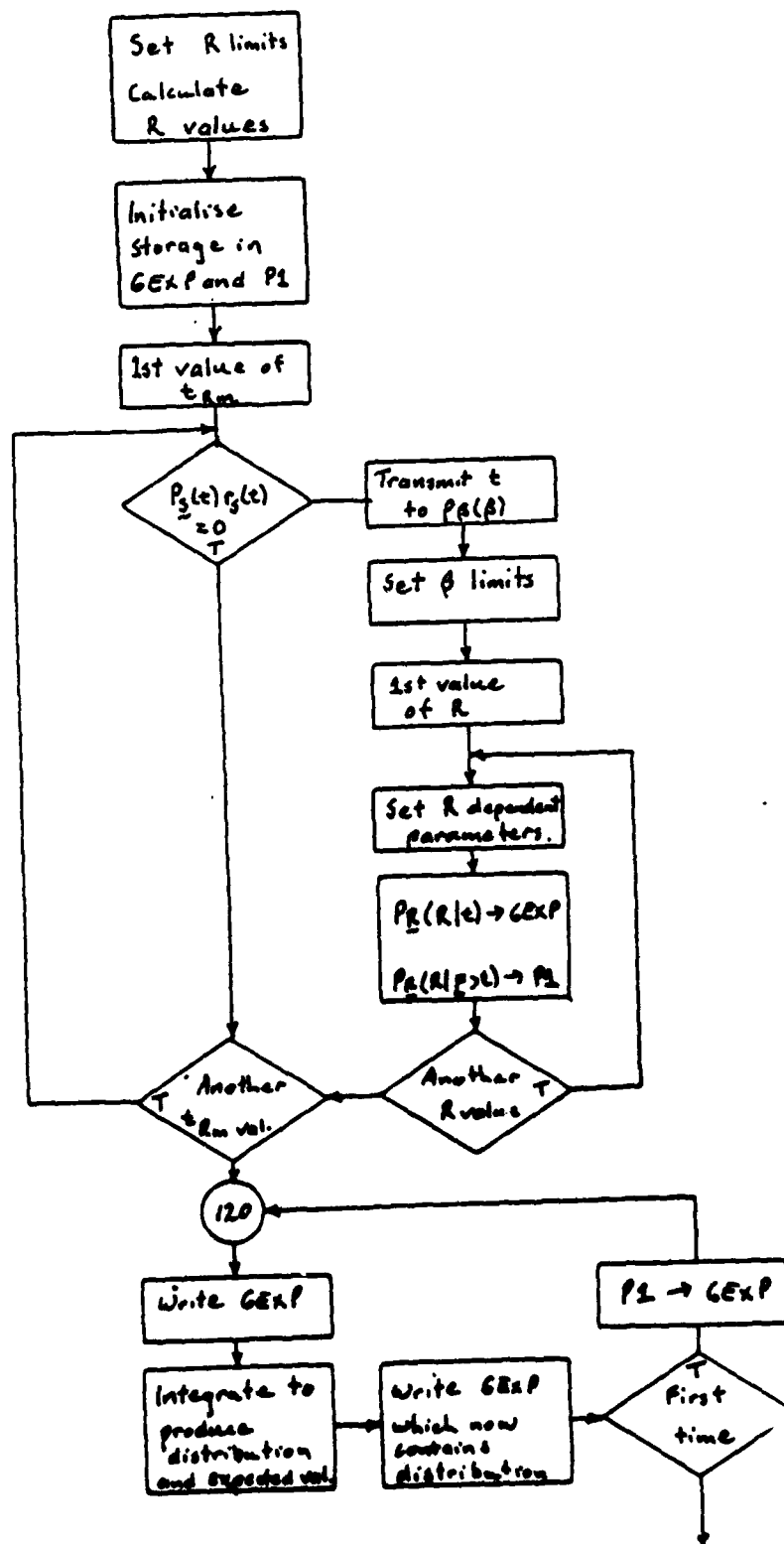


Figure 5.5. Sequence of major operations in FLPROB.

much of the code in these loops is concerned with the evaluation of the density functions and is described in Section 5.8.

Following these loops, the density function in GEXP is output to the primpay output file and to functions files for late plotting by NERPLT. Trapezoidal integration is then used to integrate each column over R to produce the cumulative distribution and the expected value. The new table containing values of the distribution is then output, together with the $Nl-1$ values of the expected value.

The contents of $P1, (p_R(R|F>t))$, are then transferred to GEXP and the operations in the penultimate paragraph repeated.

Significant Local Variables

RMIN: Minimum value for R for the evaluation of the strength functions. (Note that this minimum is not necessarily R_{min} .)

RMAX: Maximum value of R for the evaluation of the strength functions.

M1: Number of values of R.

N1: Number of columns in table. $N1=NDIST+1$ where NDIST is the number of values in $\{t_{Rm}\}$.

P1: Array storage for the table of values of the density function for strength.

GEXP: Array storage for the table of values of the density for the failing load.

T1: Storage array for residual term for the density for the failing load.

EXTNS: Array containing text strings which define the file name extensions for the function files containing the strength functions.

LABLS: Array containing text strings used to construct table headings.

Prompts

- (i) 'STRENGTH DISTRIBUTION CALCULATIONS': Notification that the strength calculations have commenced.
- (ii) 'INFORMATION ***** TIME = *****': Notification that functions for a new time value are being calculated.

5.2.4. Termination

NERF can terminate in three ways.

- (i) The sequence of operations can reach the end of PROGRAM NERF.
- (ii) The allowed run time can be exceeded, as detected by subroutine EXTIME
- (iii) An error condition can result in premature cessation of operations.

Conditions (i) and (ii) can be treated together as 'normal terminations'. In such cases, a new control file is produced by the subroutine CFNEW so that the next run can use updated information. In cases where an error condition has occurred, the control file is not updated as it is assumed that the run will recommence from the initial (or user altered) control file.

In all cases final termination is made by the subroutine FINISH which calculates the program run time before completing the run.

The subroutines CFNEW and CFOUT are described below. EXTIME and FINISH are described in Section 5.9.

SUBROUTINE CFNEW

Function

CFNEW produces an updated control file following normal termination of the operation of NERF. The new control file contains an updated run number, new inspection times and new contents for the COMMON block RISKCM.

Operation

CFNEW controls the following operations.

- (i) The original control file is read to restore the first group of data to their original values.
- (ii) The run number is decoded from the text string in TITLE, the run number incremented by one and a new text string is generated.
- (iii) CFOUT is called to output the new control file.
- (iv) The text string in TITLE is restored to its original form for use by the subroutine FINISH.

287

SUBROUTINE CFOUT

Function

CFOUT creates a new control file from the data stored in NERF.

Operation

CFOUT is actually an entry point in the subroutine CFIN and essentially mimics the input operations performed by that subroutine. CFIN outputs both groups of data.

5.2.5. Program output

The description of NERF has so far concentrated on the sequence of operations required to generate the reliability functions. In this Section the various output facilities provided by NERF are described.

The computer program NERF is, in fact, one of a suite of three programs, NERPRE, NERF and NERPLT which combine to form a complete facility for the evaluation of reliability functions for fatigue. The program NERPRE has already been referred to and provides an interactive facility for generating the control file (CONTRL.IN) and various function files which define the input functions as sequences of ordered pairs of argument and function values. NERPLT is an interactive plotting program which can plot combinations of functions defined by function files, or the probability density functions used by NERF.

The relationships between these programs and the various input and output files is shown schematically in Figure 5.6. The input data, which has already been described in Section 5.2.1., consists of the control file, the loss factor interpolation table and function files defining $a(t)$, $R(a)$, $\bar{P}_L(R)$ and $C_d(a)$. The loss factor table (GCOM.DAT) is optional and can only be produced by NERF via the subroutine SETTAE. The presence of GCOM.DAT as an input file serves only to save computational effort: it is not necessary to define the reliability calculation.

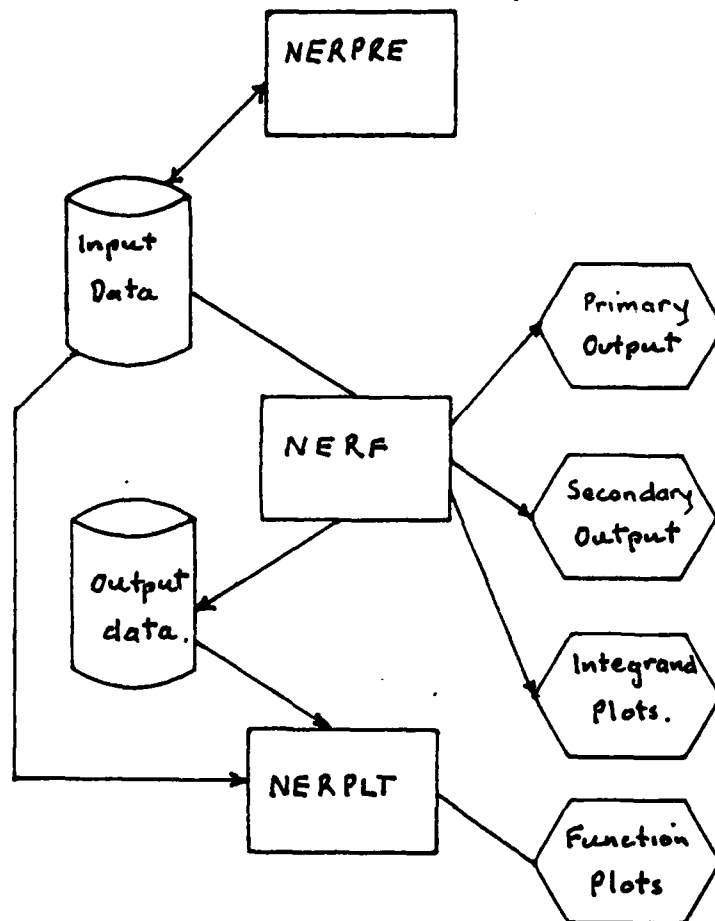


Figure 5.6. The relationships between the three programs NERPRE, NERF and NERPLT and the various input and output files.

The 'output data' (as identified in Figure 5.6) produced by NERF is in the form of function files which define the reliability functions as sets of ordered pairs. These function files have a format which is similar to that of the input function files (specified by the function READFN described in Section 5.9) but which allows several functions to be tabulated for the same set of argument values. The only program which can read these files is NERFPLT which program's function is identified in Figure 5.6 as the production of 'function plots'.

During the creation of the time sequence of reliability functions a function file containing $r(t)$, $r_s(t)$, $r_f(t)$, $r_v(t)$, $P_F(t)$, $P_S(t)$, $E(F)$ and $r_{\text{mean}}(t)$ for all the values t_k . For time values corresponding to inspections, the function $P_{\text{det}}(t)$ is also tabulated.

Similar function files are produced by the subroutine FLPROB to define the various strength distributions. Four files are produced each containing a single function for several values of time (t_{Rm}) tabulated for a range of values of R as determined automatically by NERF. The five function files produced during a single run by NERF are summarised in Table 5.7.

Note that the file names for the function all have the same 'root', v12, the title for the run. The current version of NERPRE constructs a title which consists of a two character identification followed by a three character number, (e.g. LM300). The extension identifies the function file.

File Name	Contents
IDNUM.FCN	$r(t)$ Total risk rate $r_s(t)$ Risk of static fracture $r_f(t)$ Risk of fatigue life exhaustion $r_v(t)$ Virgin risk $P_F(t)$ Probability of failure $P_S(t)$ Probability of survival $E(F)$ Expected time to failure $r_{mean}(t)$
IDNUM.FLD	$p_R(R t)$ Density for the failing load
IDNUM.FLD	$P_R(R t)$ Distribution of the failing load
IDNUM.RSD	$p_R(R F > t)$ Density for strength
IDNUM.RSP	$P_R(R F > t)$ Distribution of strength

Table 5.7. Summary of function files produced by NERF.

Note that IDNUM stands for the 5 character string identifying the run.

In addition to the function files, NERF produces 3 text records of the program operation, a primary output file containing the main results of the calculations, a secondary output file containing information regarding the numerical integrations and a dialogue on the remote terminal from which NERF is run. The last can be omitted by running NERF in the Batch mode. This can be done from the remote terminal to suppress unwanted output, or as was originally intended, to permit NERF to be run in an overnight batch stream. (The batch option is specified by setting BATCH=true in the input data; Table 5.3.)

The primary output file is the 'official record' of a run. It contains a heading, produced by the subroutine HEAD, which records all the relevant input data. The heading is followed by a tabulation of the time dependent reliability functions. This tabulation is then followed by tabulations of the strength distributions.

The secondary file contains the record of all the numerical processes used to generate the reliability functions. Each integration produces an informative prompt regarding the error estimates and number of function evaluations used. Warning and error messages issued by the various numerical algorithms also appear in the secondary output file. The secondary file is the 'operator's' record of the run.

Output to the primary output file is handled by the subroutines HEAD, OUTPUT and FLPROB together with a small section of code

in PROGRAM NERF. HEAD constructs the heading echoing the input data. OUTPUT constructs the table of time dependent reliability functions and also transmits the required information to the time sequence function file. FLPROB constructs the tables of strength distributions.

The secondary output file receives a heading from PROGRAM NERF. Otherwise all output to this file passes through the routines ARROUT, PROMPT and ECHO. ARROUT writes the contents of a two-dimensional array and is used to dump the loss factor table or two-dimensional integrand functions if requested during an interactive run. During a batch run only the loss factor table (if created) is dumped in this way. A companion subroutine IRROUT performs the same operation for integer arrays.

The subroutine PROMPT is used to write messages to the secondary file. The subroutine also transmits the messages to the remote terminal if the batch switch is off (i.e BATCH = false). The subroutine ECHO writes operator responses into the secondary output file.

Apart from a small section of code in SETTAB which writes a heading in the secondary output file prior to calling ARROUT to dump the loss factor table, the subroutines mentioned in the last three paragraphs are the only sub-programs in the NERF program which contain WRITE statements.

Output associated with PROGRAM NERF, SETTAB and FLPROB has already been described. The subroutines ARROUT, IRROUT, PROMPT

and ECHO are described in Section 5.9 as they are considered to be part of the basic communication aspects of the program. This leaves the subroutines HEAD and OUTPUT which are described below.

295

SUBROUTINE HEAD

Function

HEAD writes informative headings into the primary output file (Logical unit number 3)

Operation

Most of the code in HEAD consists of WRITE statements and appropriate FORMAT statements. The exception is the small section of code near the beginning of the subroutine which computes the loss factor term, $H(R_0, \bar{\epsilon}_f, \bar{\epsilon}_1, t)$ which is output as information regarding the significance of fatigue life limiting. If losses from the population are not neglected, then this term can be obtained from the loss factor interpolation table. Otherwise, it must be obtained by integration, using the function RINTV (see Section 5.4.2.)

SUBROUTINE OUTPUT(RNS,NP)

Function

OUTPUT controls the output of information following the evaluation of the reliability functions.

Parameter List

RNS: Value of time for which the values have been computed.

NP: Control integer.

=0: Headings are output.

=1: The time is not an inspection.

=2: The time is just after an inspection.

=3: A list of inspection times and the relevant probabilities of detection are output.

Operation

The only difference between a non inspection output and one just following an inspection is that the latter requires the probability of detection to be included in the list of variables. Subject to this comment, the operation of OUTPUT is a straightforward implementation of the above definitions.

Full heading and information are transmitted to the primary output file (logical unit number 3). More concise information is transmitted to the function file for time dependent functions, (logical unit number 2) and during the execution of the heading mode, informative messages are transmitted to the secondary output file, (logical unit number 4).

5.2.6. Graphics operations

As described in the previous Section, the program NERPLT provides a facility for constructing graphs of the reliability functions produced by NERF. Because these functions are stored in function files that have the same format as those used to define the input functions, NERPLT can also be used to plot any of the input functions. Figures 3.1 to 3.10 and 3.16 to 3.18 were produced by NERPLT.

In addition to the facilities provided by NERPLT, NERF provides the following which are available when the program is run interactively from a remote terminal.

(1) Outer integrand plots

Any of the reliability functions can be represented by equation (4.42), i.e.,

$$R = \int_{z_1}^{z_2} I_z(z) dz \quad (5.17)$$

where R represents the reliability function and z the outermost variable of integration. By plotting the values of $I_z(z)$ as the integrand is evaluated, a graph of the function can be constructed. The graph, constructed in this way, also indicates the performance of the adaptive integration algorithm in allocating the integrand evaluation locations.

This facility is available for all the reliability functions that are evaluated during the construction of the time sequence and which involve at least one level of integration.

A typical example is shown in Figure 5.7. The graph is for the outer integrand of $p_{fs}(t, n_0)$ ($\equiv P_{\underline{s}}(t)r_s(t, n_0)$) for example A. The model has no initial cracks so that $p_{fs}(t, 0)$ is given by equation (3.96) with $n_0 = 0$, i.e.,

$$p_{fs}(t, 0) = \int_{\beta_1(0)}^{\tilde{t}_f^*(R_{\min})} p_{\beta}(\beta) \int_{\alpha^*(0, \beta)}^{\alpha_2(\beta, R_{\max})} r_2(\alpha, \beta) H(\alpha, \beta, 0, t) p_{\alpha}(\alpha) d\alpha d\beta \quad (5.18)$$

which for a given value of t can be represented by,

$$p_{fs}(t, 0) = \int_{\beta_1}^{\beta_2} F_{\beta}(\beta) d\beta \quad (5.19)$$

where,

$$F_{\beta}(\beta) = \int_{\alpha^*(0, \beta)}^{\alpha_2(\beta, R_{\max})} r_2(\alpha, \beta) H(\alpha, \beta, 0, t) p_{\alpha}(\alpha) d\alpha \cdot p_{\beta}(\beta) \quad (5.20)$$

Figure 5.7. shows $F_{\beta}(\beta)$ for $t = 4000$ hrs. Note that, unlike the figures presented in Chapter 3, Figure 5.7 contains the labels actually constructed by NERPLT. Note the heading contains the run identification, UM321 (run 321 for the series identified by 'UM'). The name RSLT identifies $p_{fs}(t, n_0)$ and originated from earlier notation for the reliability functions. The

299

UM321 RSLT TERM

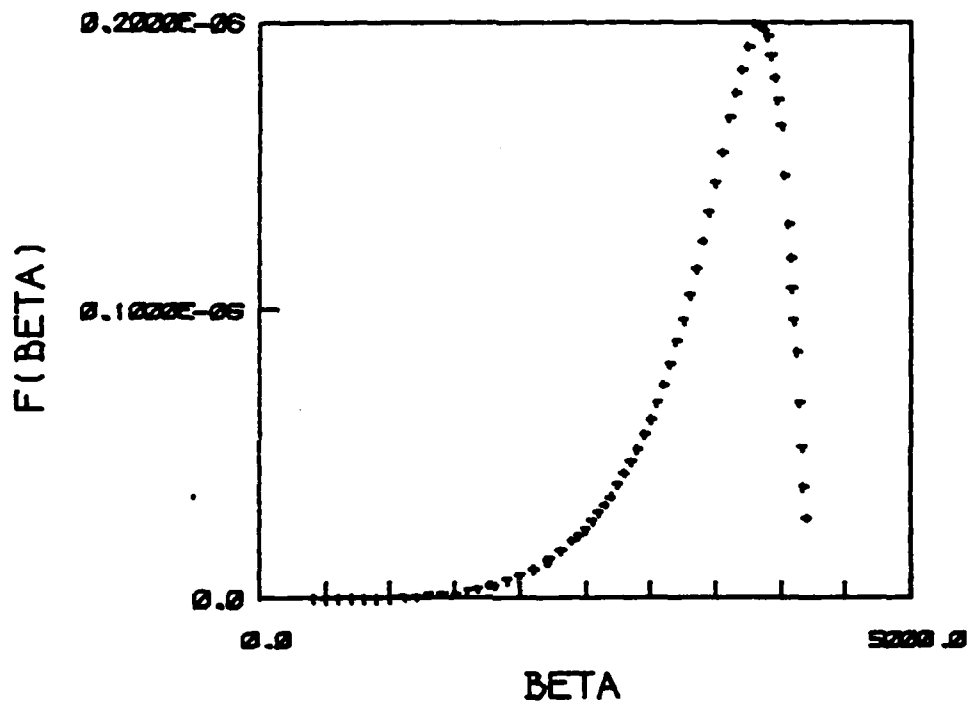


Figure 5.7. Outer integrand plot for $p_{fs}(t,0)$
for example A, $t=4000$ hrs.

equivalences between the labels used by the plotting facilities and the present notation is given in Section 5.10.

(ii) Integrand - function evaluation maps

For the most general model (Section 3.4.2) with constant n_0 , the functions $P_F(t, n_0)$ and $p_{fs}(t, n_0)$ are defined by double integrals. The integrands for these functions can be represented by two-dimensional contour maps as shown by the example given in Figure 5.8. This map is for the same integration as the outer integral plot shown in Figure 5.7. The contour lines represent values of the function,

$$I_{\alpha, \beta}(\alpha, \beta) = p_{\beta}(\beta) \cdot r_2(\alpha, \beta) H(\alpha, \beta, 0, t) p_{\alpha}(\alpha) \quad (5.21)$$

which is the integrand of equation (5.18) if the integration is regarded as a double integral rather than the nested sequence represented by equations (5.19) and (5.20).

Note that the integration domain is not rectangular and follows the boundaries defined in Section 3.3.

Figure 5.9. shows an integrand map for the same function but for $t=1500$ following an inspection at $t=1000$ hrs. The limits shown in this figure can be identified by studying Figure 3.13.

UM321 RSLT INTEGRAND

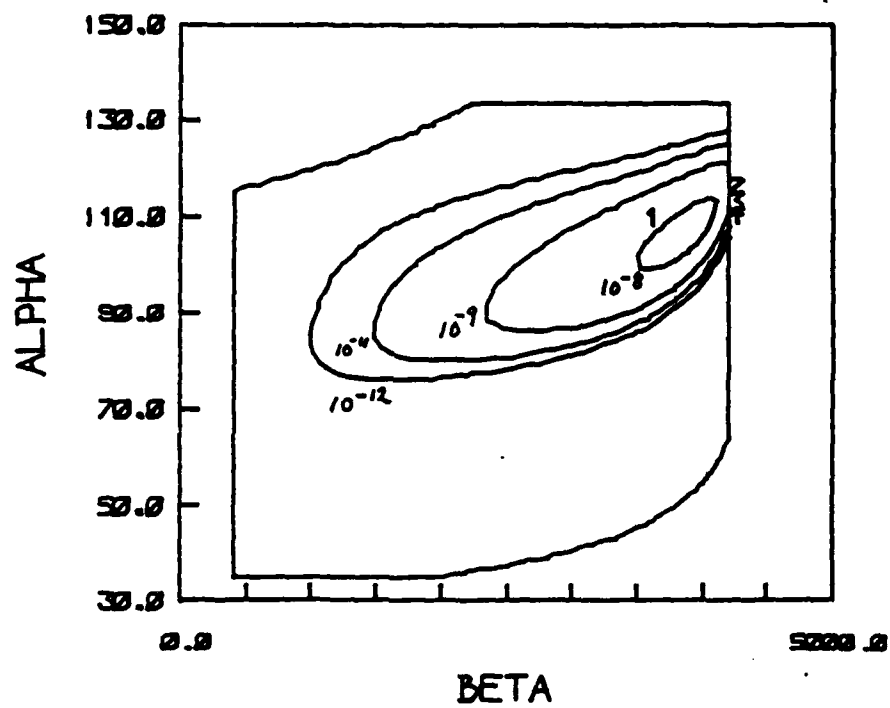


Figure 5.8. Integrand contour map for $p_{fs}(t, 0)$
for example A, $t = 4000$ hrs.

UM322 RSLT INTEGRAND

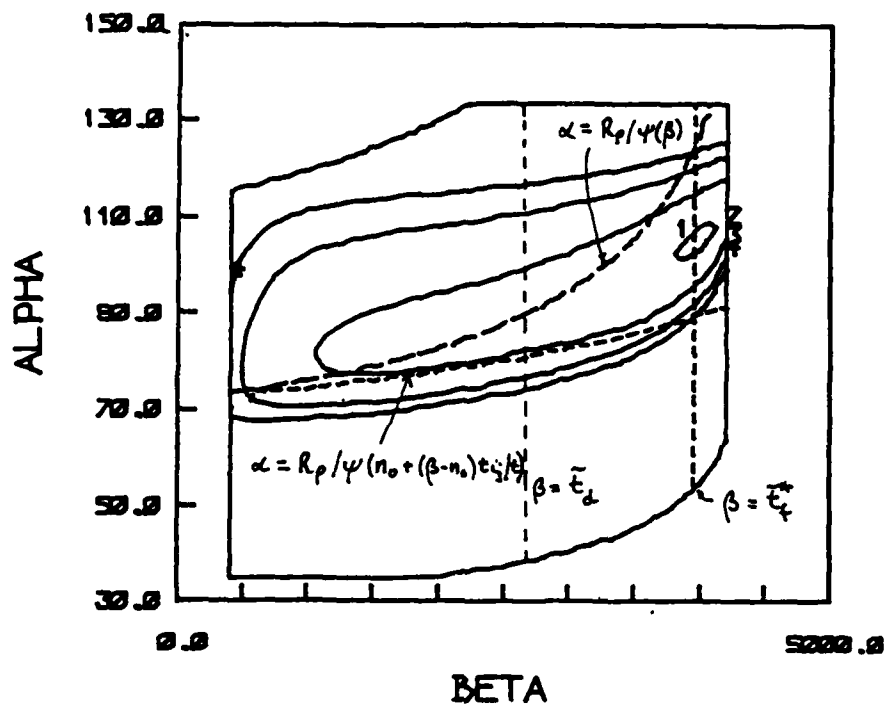


Figure 5.9. Integrant contour map for $p_{fs}(t,0)$ for example A, $t=1500$ hrs following an inspection at 1000 hrs. The broken lines identify limits arising from the inspection logic as indicated.

(iii) Loss factor maps

Contour maps in (α, β) space for the functions used to evaluate the loss factor, $H(\alpha, \beta, n_0, t)$ can also be constructed. The significance of these maps can be appreciated only after the interpolation procedures for the evaluation of the loss factor have been described. For this reason, the description of this facility is deferred until Section 5.10.

The graphics facilities described in this Section are provided by a suite of graphics subroutines which are described in Section 5.10. It is possible to load NERF with dummy routines which replace the graphics routines and remove, completely, any reference to graphics software or equipment. The graphics subroutines can be regarded as being distinct from the main body of the code associated with the evaluation of the reliability functions.

Any further description of the graphics code is left for Section 5.10. The only exceptions to this are those sections of the reliability function evaluation code which either initiate graphics operations or store numerical values for later access by the graphics routines. This code is described under the heading 'graphics operations' in the subroutine descriptions, (e.g. description of ADVNCE, p 271).

5.3. The Evaluation of Input Functions

The input functions comprise the density functions for the random variables and relationship between crack length and age ($a(\tilde{t})$), the relationship between strength and crack length ($\tilde{R}(a)$), the probability of load exceedence ($\tilde{P}_L(R)$) and the crack detection function ($C_d(a)$). Specifications for these functions were given in Section 3.2. This Section details the coding associated with their evaluation.

The functions $a(\tilde{t})$, $\tilde{R}(a)$, $\tilde{P}_L(R)$ and $C_d(a)$ are defined by sequences of ordered pairs of function and argument. These sequences are presented as input data to NERF which generates auxiliary functions that are used during the calculations. These auxiliary functions are; relative strength as a function of age,

$$\psi(\beta) = \psi(\tilde{t}) = \tilde{R}(a(\tilde{t})) / \tilde{R}_0 \quad (5.22)$$

the two risk rate equations,

$$r_1(R) = r_1(\alpha) = 1_r \cdot \tilde{P}_L(\alpha) \quad (5.23)$$

$$r_2(R) = r_2(\alpha \psi(\beta)) = 1_r \tilde{P}_L(\alpha \psi(\beta)) \quad (5.24)$$

and the inspection removal function,

$$S_j(\tilde{t}) = 1 - C_d(a(\tilde{t})). \quad (5.25)$$

As well as these auxiliary functions NERF also generates inverse and derivative functions when required.

Cubic spline interpolation, using the function FINTRP described in Section 4.2, is used to generate these functions with the function DERIV being used to generate the derivative. The inverse function is found via the secant method described in Section 4.3.

Each of the input and auxiliary functions is evaluated by its own specific function routine in the NERF package. These function routines are written in such a way that the method of evaluation, whether by interpolation or not, is concealed from the calling code. In principle, the convention of using direct evaluation for the density functions and interpolation for the other functions could be changed without any effect to the code using the functions.

All functions are initialised the first time the function is called. In some cases a function requires another to be initialised before it can be used, (e.g. the derivative functions). Where possible, the initialisation of such pre-requisite functions is automatic. Where this is not achieved the documentation describes the required initialisation sequence. (The initialisation sequence in CFIN currently satisfies all such requirements.)

The functions are described below in groupings according to the function type as defined in the input data, i.e, functions depending on $a(\tilde{r})$, $\tilde{R}(a)$, $\tilde{P}_L(R)$, $C_d(a)$ or the density functions.

5.3.1. Crack growth function

The crack growth function is specified in Section 3.2.1 and is evaluated by the function CRKGR. The derivative function is provided by CRKDEV and the inverse by CRKINV.

Initialisation is effected via the call

XXX = PSI(0.0)

in CFIN. Because the relative strength decay function requires the crack growth function initialisation of CRKGR is provided within PSI.

307

FUNCTION CRKDEV(ARG)

Function

CRKDEV evaluates the derivative of crack length with respect to age.

Parameter List

ARG: The value of age for which the derivative is required.

Operation

The derivative is evaluated by calling DERIV which is an entry point in the interpolation software. No check is made to ensure that the crack growth function has been initialised. It is up to the calling software to ensure that at least one call to CRKGR is made before calling CRKDEV.

FUNCTION CRKGR(ARG)

Function

CRKGR evaluates the crack growth function as defined by a set of ordered pairs of crack length and age.

Parameter List

ARG: The value of age, \tilde{t} , at which the crack length is required.

Operation

The first time CRKGR is called, the data defining the function is read from a function file (identified by text in CNAME in the common block FNAMES) using the subroutine REALIN. The interpolation function FINTRP is then called to initialise the interpolation tables which will be used throughout the current run of NERF as a basis for the evaluation of the crack growth function.

Evaluation of the crack length function is made by calling FINTRP using the tables stored in the common block CRKCOM.

If ARG is less than the first value in RNC (the array of argument values) the length corresponding to RNC(1) is returned.

Error Messages

- (1) 'NOT ENOUGH STORAGE FOR CRACK GROWTH FUNCTION'; The allocated space in CRKCOM is not sufficient to store the data contained in the function file. The error is fatal and the execution of NERF is terminated.

COMMON Variables Accessed

CNAME: Five character root for name of the function file containing the ordered pairs defining the crack growth function.

309

COMMON Variables Changed

(First call only.)

NCRK: Number of data pairs defining the crack growth function.

RNC: Array of ages.

CRK: Array of crack lengths.

FUNCTION CRKINV(ARG)

Function

CRKINV evaluates the age corresponding to a given crack length. It is the inverse function for CRKGR.

Parameter List

ARG : Value of crack length for which the age is required.

Operation

The function FSOLVE is used to seek a solution of the equation

$$\text{ARG} = \text{CRKGR}(\text{CRKINV}). \quad (5.26)$$

Initial guesses for CRKINV for use by FSOLVE are found by ascertaining the interval in the sequence of crack lengths which contains ARG. This is done by calling INDLOW. If ARG is very close to a node, the nodal value is returned. The resolution of this test is governed by EPS2 in COMMON block SMALCM. EPS2 is set to the order of the precision of the computer on which NERF is installed.

If ARG is less than the first crack length in CRK (the array of crack lengths), then CRKINV is returned as the corresponding age. Similarly an argument greater than the final crack length returns the corresponding age.

Note that no test is made to ensure that the crack growth function has been initialised. The external code must ensure that at least one call to CRKGR precedes a call to CRKINV.

COMMON Variables Accessed

RNC: Array of ages defining the crack growth function.

CRK: Array of crack lengths defining the crack growth function.

5.3.2. Strength decay functions

The strength decay function is specified in Section 3.2.2 and defines median strength as a function of crack length. It is evaluated by the function STRFN.

The strength decay function is used to generate the relative strength function $\psi(\tilde{\epsilon})$ which is given by equation (5.22). Given $a(\tilde{\epsilon})$ and $\tilde{R}(a)$, $\psi(\tilde{\epsilon})$ can be defined for the nodal values of $\tilde{\epsilon}$ used to define $a(\tilde{\epsilon})$.

The relative strength function is initialised during the first call to PSI which is subsequently used to provide evaluations of the function. During this initialisation dummy calls to STRFN and CRKGR are made to ensure all relevant functions have been initialised. In this way a single call to PSI in CFIN is sufficient to initialise all the functions which depend on $a(\tilde{\epsilon})$ and $\tilde{R}(a)$.

Inverse and derivative functions are provided by the functions PSINV and PSIDEV respectively.

A facility for re-establishing the interpolation tables following redefinition of the nodal values of $\psi(\tilde{\epsilon})$ is provided by PSISSET.

FUNCTION PSI(ARG)

Function

PSI evaluates the relative residual strength as a function of age. The function is defined by a set of ordered pairs and interpolation is used to provide a continuous function.

A secondary function is the definition of the initiation age, \tilde{t}_i and fatigue life limit, \tilde{t}_f for the model. This function is provided the first time PSI is called only. (Note that these parameters are, in fact, specified by the limits for the crack growth function: these limits are also taken for $\psi(\beta)$.)

The values of \tilde{t}_i and \tilde{t}_f thus defined are subject to the default limits equations applied by SETTAB. It is possible, via PSISSET, to re-establish the definition of PSI to conform to the adjusted limits.

Parameter List

ARG: Value of age for which the relative residual strength is required.

Operation

The function PSI includes entry points PSISSET and PSIDEV. The principal operations executed by the code follow the flow chart shown in Figure 5.10.

(i) Initialisation

The first time PSI is called it executes code which initialises the interpolation procedures for crack growth and the strength decay functions. The relative strength function is then initialised by generating a set of ordered pairs of age and relative residual strength, ψ , by evaluating equation (5.22) for each value of β used to define the crack growth function $a=a(\tilde{t})$. The set of ordered pairs thus generated are then truncated to ensure that $\psi(\beta) \leq 1$.

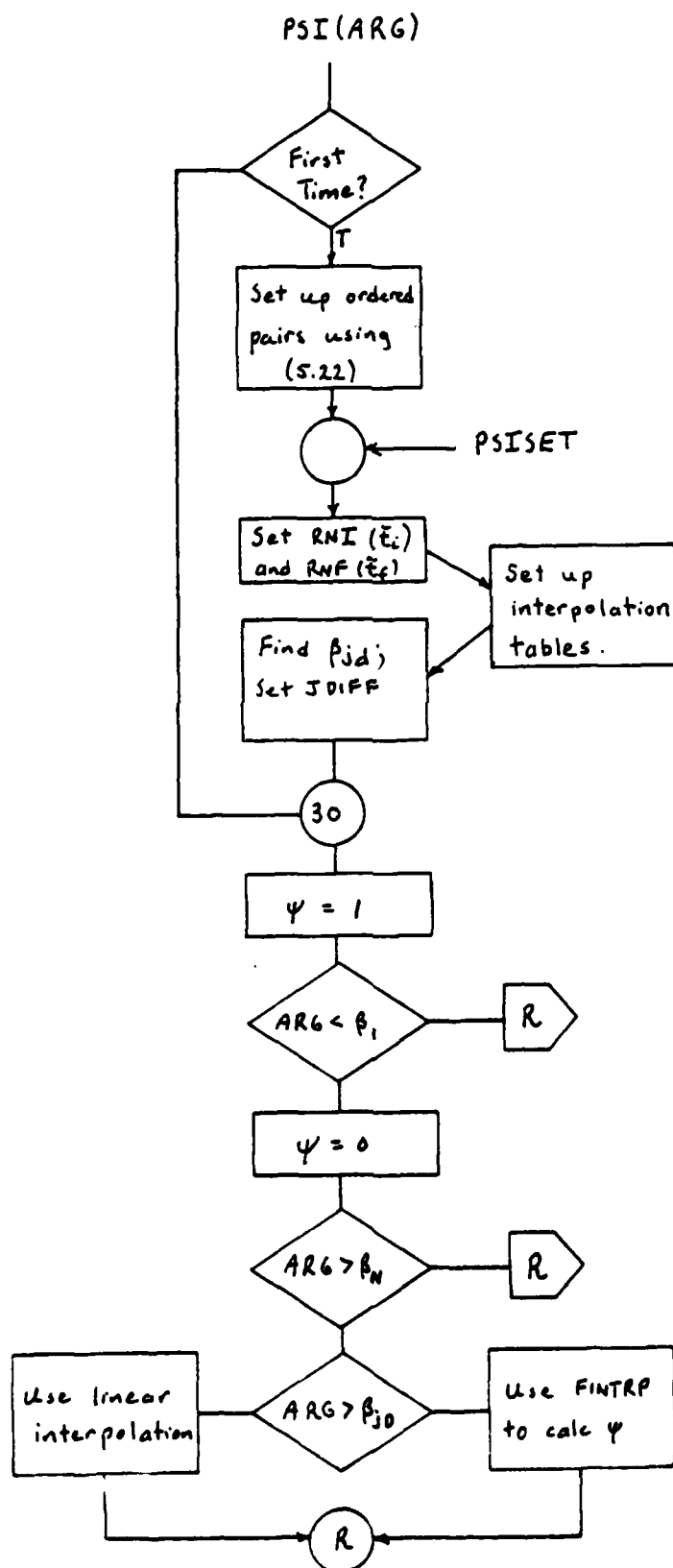


Figure 5.10. Logic within FUNCTION PSI associated with the evaluation of $\psi(A)$.

Having established an ordered pair sequence defining $\psi(\beta)$, the interpolation tables are initialised by calling FINTRP (see Section 4.2.2). The sequence of age values defining $\psi(\beta)$ are also used to define $\tilde{\tau}_1$ and $\tilde{\tau}_f$. Note that $\tilde{\tau}_1$ is set to be at least equal to a small positive number so that the condition $\beta=0$ can be interpreted as being relevant for uncracked structures for which PSI will return the value 1. The corresponding variables (RNI and RNF) in COMMON are set by this section of the code which can also be accessed at any time via the alternative entry point, PSISSET. This alternative entry can be useful if external code modifies the ordered pair sequence defining $\psi(\beta)$.

The interpolation function for $\psi(\beta)$ is linear for $\tilde{\tau}_1 \leq \beta < \beta_{jd}$ where β_{jd} is the largest value of the β sequence defining $\psi(\beta)$ for which $\psi(\beta) \geq 0.9$. The final step in the initialisation process is to use the subroutine INDLOW to find the location of this node in the sequence and to save β_{jd} and the corresponding ψ value, ψ_{jd} in the variables BETJD and PSIJD respectively.

(11) Evaluation

Following initialisation, evaluation of $\psi(\beta)$ is simply a matter of using the appropriate interpolating function. If $\beta < \beta_{jd}$, linear interpolation is made using code within PSI. Otherwise FINTRP is used to use cubic spline interpolation via the tables previously initialised.

If β is less than the first nodal value, $\tilde{\tau}_1$, ψ is returned with the value 1. If β is greater than the largest β node, $\tilde{\tau}_f$, the value 0 is returned. Note that the function $\psi(\beta)$ thus defined can be discontinuous at each end of the interval $[\tilde{\tau}_1, \tilde{\tau}_f]$.

COMMON Variables Changed

BETA: Array of β values defining $\psi(\beta)$.

BETJD: β_{jd} , or β value marking the transition from linear interpolation to cubic spline interpolation.

JDIF: The number of the node in the β sequence defining $\psi(\beta)$ corresponding to β_{jd} .

N: Number of ordered pairs defining $\psi(\beta)$.

PS: Array of ψ values defining $\psi(\beta)$.

PSJD: ψ_{jd} or ψ value corresponding to β_{jd} . See BETJD above.

RNI: Current value of \bar{t}_1 .

RNF: Current value of \bar{t}_f .

FUNCTION PSIDEV(ARG)

Function

Evaluate the value of the derivative of $\psi(\beta)$ with respect to β .

Parameter List

ARG: Value of β for which the derivative is required.

Operation

PSIDEV evaluates the derivative of $\psi(\beta)$ either by using the derivative of a linear interpolation equation for $\beta \leq \beta_{jd}$, (where β_{jd} is defined by the function PSI) or accessing DERIV which uses cubic spline interpolation.

If β is not in the interval $[\tilde{\epsilon}_1, \tilde{\epsilon}_f]$, the value 0 is returned for the derivative.

Note that the interpolation tables must have been initialised by a previous call to PSI before PSIDEV is called. PSIDEV is an entry point in PSI and can therefore detect if such a call has been made. If the tables are not initialised an error message is created and the operation of NERF is terminated. (Note that this error can only occur if the NERF computer program is changed from its form at the time of documentation.)

Error Message

- (1) 'PSI DEV ... NO DATA'; PSIDEV has been called prior to a call to PSI. The interpolation tables have not been initialised.

FUNCTION PSINV(ARG)

Function

PSINV evaluates the age corresponding to a given value of relative residual strength.

Parameter List

ARG: Value of relative strength, ψ , for which age, β , is required.

Operation

The function FSOLVE is used to seek a solution of the equation

$$\text{ARG} = \text{PSI}(\text{PSINV}). \quad (5.27)$$

Initial guesses for PSINV are found by ascertaining the interval in the sequence of values of ψ defining $\psi(\beta)$ which contains the value of ψ defined by ARG. This is done by calling INDLOW. If this interval is such that linear interpolation is used to define $\psi(\beta)$ then the inverse function can be evaluated directly. Otherwise FSOLVE is used to find the inverse function.

Note that PSI must be called at least once before PSINV.

FUNCTION PSISSET(ARG)

Function

PSISSET re-initialises the interpolation tables used to define $\psi(\beta)$ following alteration of the sequence of ordered pairs of ψ and β by code external to function PSI.

Parameter List

ARG: Value of β for which PSISSET will return a value of ψ .

Operation

PSISSET is an entry point in the function PSI and its operation is identical to that described for the initialisation phase of PSI with the exception that no attempt is made to redefine the strength decay function ($\tilde{R}(a)$) or to compute new values of ψ as stored in the array, PS.

Common Variables Changed

See function PSI.

FUNCTION STRFN(ARG)

Function

STRFN evaluates the strength decay function, $\tilde{R}=\tilde{R}(a)$.

Parameter List

ARG: Value of crack length, a , for which the strength, \tilde{R} is required.

Operation

The first time that STRFN is called the data defining the strength decay function is input from disk using the subroutine READFN. The file is identified by the text string in PNAME in the COMMON block FNAMEs.

After the data from disk has been transferred to local storage, FINTRP is called to initialise the interpolation tables.

Interpolation is effected using FINTRP in a conventional manner. If the argument, ARG, is out of range, STRFN returns the limit values as defined by the first or last ordered pairs appropriately.

Storage Conflict

Note that STRFN is normally used during the initial phases of NERF only. For this reason, the ordered pairs and interpolation tables are stored in the common block GCOM which is used eventually to store the loss factor interpolation tables. The strength decay function can therefore be accessed only prior to the first call to the subroutine SETTAB which initialises the loss factor interpolation table.

5.3.3. Risk rates

The risk rate for structures of given strength R , is given by,

$$r_k(R) = \bar{P}_L(R) \cdot 1_r \quad (5.28)$$

where $\bar{P}_L(R)$ is the probability of load exceedence. The specifications for $\bar{P}_L(R)$ and $r_k(R)$ are given in Section 3.2.2.

The risk rates are evaluated by the function RLOAD which uses cubic spline interpolation to interpolate $\log(r_k(R))$.

Note that RLOAD(R) returns a risk rate that is independent of whether a structure is cracked or uncracked, (i.e. $k=1$ or $k=2$ respectively). It is up to the calling code to make the correct interpretation.

A facility for re-establishing the interpolation tables is provide by RLOSET.

FUNCTION RLOAD(ARG)

Function

RLOAD evaluates the local risk rate for structures with a given value of R. Because the argument is R, the risk rate is relevant for cracked or uncracked structures.

A secondary function, provided during the first call to RLOAD is to define the limits for strength R_{\min} and R_{\max} . These are assumed by the rest of the code in NERF to be the first and last values of the sequence of argument values defining the risk rate function.

Parameter List

ARG: Value of strength, R, for which the risk rate is required.

Operation

On initial entry, the sequence of ordered pairs defining $\bar{P}_k(R)$ is input from the function file identified by the text string in RNAME. The nodal values of $\bar{P}_L(R)$ are converted to values of $\log(r_k(R))$ before the tables are initialised.

The evaluation of $r_k(R)$ for given R is straightforward. After interpolation of the logarithmic function, the exponential of the result is taken to yield the required risk rate.

If $R < R(1)$, r_k is set to $r_k(R(1))$. If $R > R(M)$, (where M is the number of nodes in the sequence defining the risk rate function), r_k is set to 0.

FUNCTION RLOSET(ARG)

Function

RLOSET re-establishes the interpolation tables following a re-definition of the ordered pair sequence defining the risk rate function.

Parameter List

ARG: Value of R for which a value of the risk rate will be returned following re-establishment of the interpolation table.

Operation

RLOSET is, in fact, an entry point in RLOAD and commences operation at the point where RLOAD has just input the sequence of ordered pairs and calculated the values of $\log(r_k(R))$.

The facility is provided to cope with the requirement of imposing the default limit equations for R_{\min} and R_{\max} by SETTAB.

5.3.4. Inspection removal and crack detection functions

The inspection removal and crack detection functions were specified in Section 3.2.4. The crack detection function is defined by a set of ordered pairs of crack length (a) and $C_d(a)$. These data form part of the input data for NERF which converts the crack detection function to the inspection removal function $S_j(\tilde{t})$ by applying the equation (3.30), i.e.,

$$S_j(\beta) = S_j(\tilde{t}) = 1 - C_d(a(\tilde{t})). \quad (5.29)$$

The β nodes used to define $S_j(\tilde{t})$ are the same as those used to define $a(\tilde{t})$.

The function $S_j(\tilde{t})$ is used to evaluate the modified density for β , $p_\beta^*(\beta)$, given by equation (2.55),

$$p_\beta^*(\beta) = p_\beta(\beta) \cdot \left\{ \prod_{j=1}^r [S_j(n_0 + (\beta - n_0) t_{1j}/t)] \right\} \quad (5.30)$$

where r denotes the last inspection before the current time, t .

The product term in (5.30) is evaluated by the function SINSF so that (5.30) is replaced by,

$$p_\beta^*(\beta) = p_\beta(\beta) \times \text{SINSF}(\beta). \quad (5.31)$$

The function SINSF is written in the same way as the function routines for the other input functions in that initialisation occurs during the the first call to the function. . Initialisation consists of reading the function file for $C_d(a)$ and establishing a sequence of ordered pairs defining $S_j(\tilde{t})$. The largest value of a used to define $C_d(a)$ is taken

as a definition for a_d and subsequently for \tilde{t}_d , via,

$$\tilde{t}_d = a^{-1}(a_d). \quad (5.32)$$

These values for a_d and \tilde{t}_d are used by the inspection logic in the usual way.

If a non zero value has been specified for a_d prior to the initial call to SINSP, then the crack detection function is ignored and the inspection procedure is assumed to be perfect.

FUNCTION SINSP(ARG)

Function

SINSP evaluates the factor needed to correct the density function for β for the effects of all preceding inspections.

Parameter List

ARG: Value of β for which the factor is required.

Operation

On initial entry, the function is initialised by the following sequence of operations.

- (i) If CND is non zero, (i.e. a_d has already been specified) the logical variable SINGLE is given the value true to signify that the inspection removal function will be ignored.
- (ii) The crack detection function is read from the function file identified by the name stored in DNAME in the common block FNAMEs. Values of a are stored in CRPD and values of $C_d(a)$ are stored in PDFN. Both these arrays are allocated space in the common block GCOM. It is therefore essential that SINSP is called at least once before SETTAB sets up the loss factor table.

- (iii) Interpolation tables are initialised for $C_d(a)$.
- (iv) RND is set to correspond to the crack length given by the last node in CRPD, (equation 5.32).
- (v) Using interpolation equation (5.29) is applied to set up a sequence of ordered pairs defining $S_j(\tilde{t})$. The nodes correspond to the β nodes used to define $\psi(\beta)$ and are vignettted to lie in the interval $[\tilde{t}_1, \tilde{t}_d]$. The \tilde{t} values are stored in SBET and the values of $S_j(\tilde{t})$ in SFM. These arrays are located in the common block DETCOM.
- (vi) The interpolation tables for $S_j(\tilde{t})$ are initialised.

Following initialisation, the product term in equation (5.30) is evaluated if inspections precede the current time. Otherwise SINSP returns the value 1.

5.3.5. Density functions

Although the reliability models and input data are defined in terms of three basic random variables, comparative fatigue life, initial crack length and relative residual strength (\underline{X}_1 , \underline{X}_2 and \underline{X}_3 respectively), the numerical analysis is in terms of the transformed random variables, virgin strength, age and initial crack length (or initial age).

The evaluation of the density functions for the transformed random variables is handled by a total of 10 function routines which have been structured so that the processing of the transformations and the evaluation of the particular types of density functions have been separated. The details of the transformations are a characteristic of the reliability modelling and will change only when the fundamental modelling for NERF is changed. It is far more likely that the types of density functions will be modified and with the present program structure, this can be effected by changing only two of the density function routines.

The density functions are defined for a standard random variable \underline{Z} , which is related to the basic random variables by an equation of the form,

$$\underline{Z} = (\underline{X} - e)/v \quad (5.33)$$

where v and e are called the location and minimum value parameters respectively. It follows that

$$p_{\underline{X}}(x) = p_{\underline{Z}}((x-e)/(v-e))/(v-e). \quad (5.34)$$

The relationship (5.33) can be generalised by introducing a third random variable \underline{Y} , related to \underline{X} by,

$$\underline{Y} = f_s \underline{X} \quad (5.35)$$

where f_s is a scale factor. The density function for \underline{Y} is given by,

$$\begin{aligned} p_{\underline{Y}}(y) &= p_{\underline{Z}}((y-f_s e)/(f_s v-f_s e))/(f_s v-f_s e) \\ &= p_{\underline{Z}}((y-e_s)/(v_s-e_s))/(v_s-e_s) \end{aligned} \quad (5.36)$$

where v_s and e_s are the scaled location and minimum value parameters respectively. Obviously the effect of the scaling transformation can be incorporated within the parameters of the density function, rather than in a transformation equation which must be processed every time the density function is evaluated.

The standard density functions evaluated by NERF are, lognormal, extreme value and gamma and are given, respectively, by;

$$p_{\underline{Z}}(z) = C \cdot \exp\{-0.5(\log(z)/\sigma)^2\} \quad (C = 1/\sqrt{2\pi} \sigma) \quad (5.37)$$

$$p_{\underline{Z}}(z) = C \cdot z^{k-1} \cdot \exp\{-z^k\}, \quad (C = k) \quad (5.38)$$

$$p_{\underline{Z}}(z) = C \cdot z^{k-1} \cdot \exp\{-z\}. \quad (C=1/\Gamma(k)) \quad (5.39)$$

These standard functions are evaluated by the function PDF which incorporates the transformation (5.35) by evaluating

$$p_{\underline{Y}}(y) = C \cdot p_{\underline{Z}}(y-e_s)/(v_s-e_s) \quad (5.40)$$

where C now includes the normalisation factors together with the factor $1/(x_s - e_s)$ in equation (5.36).

The code calling PDF must retain the Parameters k , $1/(v_s - e_s)$, e_s and C , which given the distribution type and k , e and v can be computed to initialise the density function. This facility is provided by the function PDFSET.

The way PDF and PDFSET are used to generate the density functions for the transformed random variables is outlined below. Details of the coding are given in the function routine descriptions which follow.

(i) Density function for virgin strength, $p_\alpha(\alpha)$

Virgin strength, α , is related to relative residual strength \underline{x}_3 by equation (2.29), i.e.,

$$\alpha = \underline{x}_3 \tilde{R}_0 \quad (5.41)$$

so that

$$p_\alpha(\alpha) = p_{\underline{x}_3}(\alpha/\tilde{R}_0)/\tilde{R}_0 \quad (5.42)$$

which corresponds to the transformation (5.35) with $f_s = \tilde{R}_0$. The function $p_\alpha(\alpha)$ is evaluated by PALPHA and initialised by ALPSET.

(ii) Density function for age, $p_\beta(\beta)$

Age, β , is related to relative fatigue life, \underline{x}_1 and initial age, n_0 by equation (2.28), i.e.,

$$\beta = t/\underline{x}_1 + n_0 \quad (5.43)$$

so that given n_0 and t ,

$$p_\beta(\beta) = p_{\underline{x}_1}(t/(\beta - n_0))t/(\beta - n_0)^2 \quad (5.44)$$

which can be written in the form

$$p_{\beta}(\beta) = p_{\underline{Y}}(1/(\beta - n_0))/(\beta - n_0)^2 \quad (5.45)$$

where

$$p_{\underline{Y}}(y) = p_{\underline{X}_1}(y \cdot t) \cdot t. \quad (5.46)$$

$p_{\underline{Y}}(y)$ is the density

function returned by PDF with $f_3 = 1/t$.

The transformation from \underline{Y} to β is handled by the function PBETA which evaluates the modified density $p_{\beta}^*(\beta)$ which incorporates the effects of the removal function associated with crack length inspections.

PBETA is supported by the function BETSET which initialises the density function and BETCNG which allows the scale factor to be set to a new value of $1/t$.

(iii) Density function for initial crack length, p_{a_0}

The density function for crack length is defined directly by the input data as no transformation is involved. The functions PRNO and RNOSET provide for evaluation and initialisation. A third function, RNONRM allows for normalisation of the density function (i.e. 3.33) via modification of the constant C in equation (5.40).

(iv) Initialisation and range limiting

The three density functions for the transformed random variables are initialised by subroutine CFIN. Each initialisation (using the appropriate initialisation function) is followed by an application of RANGE, which is described in Section 4.4.1. to find the viable limits for the random variable. For initial crack length, an integration of the density function between the limits (which may be vignettted by those specified in the input data) is made to obtain a normalising factor.

FUNCTION ALPSET(K1,ALP1,ALP2,ALP3)

Function

ALPSET initialises the density function for virgin strength, α , given the parameters defining the density function for relative residual strength, \underline{X}_3 .

Parameter List

K1: Integer defining the type of density function (see PDF).

ALP1: Dispersion parameter for \underline{X}_3 .

ALP2: Lower limit parameter for \underline{X}_3 .

ALP3: Location parameter for \underline{X}_3 .

Operation

The transformation from \underline{X}_3 to α involves the use of the scale factor \hat{R}_0 , which is stored in RMU0 in the COMMON block RMU0.

A heading is output on the primary output file so that information produced by PDFSET can be correctly identified.

FUNCTION BETCNG(RNSV)

Function

BETCNG allows the scale factor used for the evaluation of $p_{\theta}(\beta)$ to be changed, according to the time value stored in RNSVIN.

Parameter List

RNSVIN: Value of t which is to be used to set a new scale factor.

Operation

PDFSET is used to re-initialise the density function $p_Y(y)$ which used to generate $p_{\theta}(\theta)$ in PBETA.

If the scale factor is changed, the viable limits for β will also change. Rather than call RANGE every time that a new time value is used, the limits are calculated approximately by multiplying the original estimates (produced with $t=1$) by t .

FUNCTION BETSET(K1,BET1,BET2,BET3)

Function

BETSET initialises the density function for age, β , given parameters defining the density function for relative fatigue life, X_1 .

Parameter List

K1: Integer identifying the type of density function (see PDF).

BET1: Dispersion parameter for X_1 .

BET2: Lower Limit parameter for X_1

BET3: Location parameter for X_1 .

Operation

The density function is initialised with $t=1$ so that no scale factor is involved. This ensures that the limits which are computed following the first call to BETSET can be used by BETCNG to compute approximate limits for other values of t .

Note that output to the primary output file is suppressed following the first call to BETSET.

FUNCTION PALPHA(ARG)

Function

PALPHA evaluates the density function for virgin strength,
 α .

Parameter List

ARG: Value of α for which the density function is to be
evaluated.

Operation

PALPHA makes direct use of the function PDF to evaluate the
density which is assumed to be initialised by a previous call to
ALPSET so that the appropriate constant are available.

FUNCTION PBETA(ARG)

Function

PBETA evaluates $p_{\beta}^*(\beta)$, the density function for age.

Parameter List

ARG: Value of β for which the evaluation of the density function is required.

Operation

PBETA accesses RNO in the COMMON block PARCOM to compute the value of \underline{y} where,

$$\underline{y} = 1/(\beta - n_0) = 1/(ARG - RNO) \quad (5.47)$$

which is passed into PDF to compute $p_{\underline{y}}(y)$. The required density for β is then given by,

$$\begin{aligned} p_{\beta}(\beta) &= \underline{y}^2 p_{\underline{y}}(y) \times S_j(n_0 + (\beta - n_0) t_{1j}/t) \\ &= ARG1 * ARG1 * PDF(KBET, C1, C2, C3, C4, ARG1) * SINSP(ARG) \end{aligned} \quad \dots (5.48)$$

The value of $\beta - n_0$ must, of course, be non-zero. An error message is produced if this condition is violated.

Error Message

- (1) 'PBETA ... ARG IS ZERO': PBETA has been called for conditions leading to $\beta - n_0 = 0$. Further processing is terminated.

FUNCTION PDF(K,C1,C2,C3,C4,ARG)

Function

PDF evaluates a probability density function, identified by the first 5 parameters in the call, for the value ARG of the random variable \underline{X} .

Parameter List

K: Integer identifying the type of distribution.

-1: Lognormal

-2: Extreme value

-2: Gamma

C1: Dispersion Constant.

C2: Scaled minimum limit.

C3: Reciprocal of scaled location parameter.

C4: Constant incorporating the effects of normalisation and scaling.

ARG: Value of the random variable for which the density function is to be evaluated.

Operation

It assumed (but in no way checked) that prior to calling PDF, PDFSET has been called to calculate the constants C1, C2, C3 and C4 which are stored by the calling code to define the density function.

The value, ARG, of the random variable \underline{X} is converted by PDF to a standard random variable, \underline{Z} say where

$$Z = (ARG - C2) * C3. \quad (5.49)$$

The density function for X can be evaluated using a standard density function for Z as described in Section 5.3.4. Currently, PDF recognises three standard distributions.

(i) lognormal

$$\begin{aligned} p_{\tilde{X}}(x) &= p_Z(z) / (f_s(\sqrt{-e}) \\ &= \exp\{-0.5 * (\log(z)/\sigma)^2\} / (f_s \sqrt{-e} \sigma z) \\ &= C4 * \text{EXP1}(-0.5 * (\text{ALOG}(Z)/C1)**2) / Z \end{aligned} \quad (5.50)$$

(ii) Extreme Value

$$\begin{aligned} p_{\tilde{X}}(x) &= k * z^{k-1} \exp\{z^k\} / (f_s(\sqrt{-e})) \\ &= C4 * T1 * \text{EXP1}(-T1) / Z \end{aligned} \quad (5.51)$$

where

$$T1 = z^k = Z**C1. \quad (5.52)$$

(iii) Gamma

$$\begin{aligned} p_{\tilde{X}}(x) &= z^{k-1} \exp\{-z\} \\ &= C4 * (Z**(C1-1.0)) * \text{EXP1}(-Z). \end{aligned} \quad (5.53)$$

FUNCTION PDFSET(K,FACT,CIN1,CIN2,CIN3,C1,C2,C3,C4,IOUT)

Function

PDFSET initialises a probability density function by calculating constants that can be used by PDF to generate the required function using standard density functions.

Parameter List

K: Integer identifying the type of density function.

-1: Lognormal

-2: Extreme value

-3: Gamma

FACT: Scale factor for the random variable X .

CIN1: Dispersion constant, defined in terms of X .

CIN2: Minimum value, defined in terms of X .

CIN3: Location parameter, defined in terms of X .

C1: Dispersion parameter for Z (Returned by PDFSET).

C2: Scaled minimum value (Returned by PDFSET).

C3: Reciprocal of scaled location parameter (Returned by PDFSET).

C4: Constant incorporating the effects of scaling and normalisation. (Returned by PDFSET)

IOUT: Integer defining output mode.

-0: Write information about the density function on the primary output file (Logical unit 3).

-1: Don't write the information.

Operation

The first section of code in PDFSET calculates the constants C1, C2 and C3 according to the equations

$$C1 = CIN1 = k \quad (5.54)$$

$$C2 = ef_s = CIN2*FACT \quad (5.55)$$

$$C3 = 1/(f_s(v - e))$$

$$= 1.0/(CIN3*FACT - C2) \quad (5.56)$$

The remainder of the operations are specialised according to the type of density function being initialised.

(i) Lognormal

For the lognormal density it is assumed that the dispersion parameter, k, has been specified in terms of logarithms to the base 10. Conversion to natural logarithms is effected by multiplying CIN1 by $1.0/\log_{10}(e)$.

The normalisation constant is given by

$$C4 = (1.0/\sqrt{2\pi}) * (1/(v_s - e_s))/k$$

$$= C3*CON/C1 \quad (5.57)$$

(ii) Extreme value

For the extreme value density function, the only specialised operation is the calculation of C4, which is given by

$$\begin{aligned}
 C4 &= k/(v_s - e_s) \\
 &= C1 * C3.
 \end{aligned}
 \tag{5.58}$$

(iii) Gamma

The normalisation constant for the gamma density function involves the evaluation of $\Gamma(k)$. This evaluation is made via the function S14AAF from the NAG library of scientific subroutines. (This is the only function which is not supplied as part of the NERF program.) Once this function has been evaluated and stored in GAM the normalisation constant is given by

$$\begin{aligned}
 C4 &= (1/\Gamma(k)) \cdot (1/(v_s - e_s)) \\
 &= C3/GAM.
 \end{aligned}
 \tag{5.59}$$

Note that provision has been made for the trapping of error codes returned by S14AAF and that the Gamma function is evaluated only when CIN1 changes.

In each case appropriate information regarding the density function that has been initialised is transmitted to the primary output file. If IOUT=1, this output is suppressed.

Error Message

- (1) 'ERROR IN GAMMA FUNCTION, IFAIL = *****': The gamma function routine supplied by the NAG library has failed. For interpretation of the error code refer to the appropriate NAG documentation.

FUNCTION PRNO(ARG)

Function

PRNO evaluates the density function for initial crack length, a_0 .

Parameter List

ARG: Value of a_0 for which the evaluation is required.

Operation

As no transformations are involved, the evaluation of $p_{a_0}(a_0)$ is made by a straightforward call to the function PDF.

FUNCTION RNONRM(FACT)

Function

RNONRM modifies the constant C4 used to evaluate $p_{a_0}(a_0)$ to include the effect of a normalising factor produced by the calling code.

Parameter List

FACT: Normalising factor.

Operation

RNONRM effects the modification by changing the constant C4 which has been previously computed by PDFSET.

FUNCTION RNOSET(K1,A1,A2,A3)

Function

RNOSET initialises the density function for initial crack length, a_0 .

Parameter List

K1: Integer defining the type of density function (see PDF).

A1: Dispersion parameter for a_0 .

A2: Minimum value parameter for a_0 .

A3: Location parameter for a_0 .

Operation

There are no transformations involved. The density is initialised by a straightforward call to PDFSET.

5.4. The Evaluation of the Loss Factor

The reliability expressions involve the evaluation of the loss factor term, $H(\alpha, \beta, n_0, t)$, which is defined, for cracked structures, by equations (2.45) and (2.44) for initial cracking and no initial cracking respectively. Both equations involve an integration over β : because the loss factor appears in the innermost integration of each reliability function considerable savings in computational effort can be made if an interpolation table can be employed to bypass the integration step.

In many practical calculations, the loss factor term is very close to 1.0, (which provided the original motivation for the option for neglecting the term altogether, see Section 3.6.3) and the expense of using full integrations for every evaluation is not justified. The interpolation procedures described in this section provide a good compromise between accuracy and computational effort.

Although the loss factor is time dependent, indicating that an interpolation table may require initialisation for each value of time, it is possible to separate the time dependence from an integral term which may be represented by an interpolation table which is valid for the whole run through the time sequence. Moreover, it is possible to remove the dependence on n_0 from the table which means that a given interpolation table may be used for a wide range of calculations using the same residual strength and probability of load exceedence functions.

The separation of the time dependence is effected by writing the loss factor in the form,

$$H(\alpha, \beta, n_0, t) = \exp\{-t.G(\alpha, \beta, n_0)\} \quad (5.60)$$

where, for the initial cracking class,

$$G(\alpha, \beta, n_0) = \frac{1}{\beta - n_0} \int_{n_0}^{\beta} r_2(\alpha, \beta') d\beta' \quad (5.61)$$

and for the no initial cracking class,

$$G(\alpha, \beta, 0) = \frac{1}{\beta} \left[\int_{\tilde{\epsilon}_i}^{\beta} r_2(\alpha, \beta') d\beta' + r_1(\alpha) \tilde{\epsilon}_1 \right] \quad (5.62)$$

The dependence on n_0 is separated by defining

$$G^*(\alpha, \beta) = \int_{\tilde{\epsilon}_i}^{\beta} r_2(\alpha, \beta') d\beta' + [\tilde{\epsilon}_1 \cdot r_1(\alpha)] \quad (5.63)$$

so that

$$G(\alpha, \beta, n_0) = \frac{1}{\beta - n_0} [G^*(\alpha, \beta) - G^*(\alpha, n_0)] \quad (5.64)$$

The term in square brackets in (5.63) is included only when $n_0=0$, for which case $G^*(\alpha, n_0)$ in (5.64) is zero. With these conventions, equations (5.63) and (5.64) are valid for all classes of models.

For uncracked structures, the loss factor is given by

$$H(\alpha, \beta, n_0, t) = \exp(-r_1(\alpha)t) \quad (5.65)$$

No integration is required and the term can be evaluated either directly within the code which evaluates the reliability functions, or as a special case of the code used to evaluate $G(\alpha, \beta, n_0)$. The latter facility is provided within the function GVAL, described below, although there are some instances in the NERF code where the loss factor term for uncracked structures is evaluated directly. This special case is identified by GVAL if $n_0=0$ and $\beta = 0$.

Another special case occurs in the initial cracking model when $\beta - n_0 = 0$. Considering the expression for $G(\alpha, \beta, n_0)$ in (5.61)

$$\lim_{\beta \rightarrow n_0} G(\alpha, \beta, n_0) = \lim_{\beta \rightarrow n_0} \frac{1}{\beta - n_0} \int_{n_0}^{\beta} r_2(\alpha, \beta') d\beta'$$

$$= \frac{\lim_{\beta \rightarrow n_0} \int_{n_0}^{\beta} r_2(\alpha, \beta') d\beta'}{\lim_{\beta \rightarrow n_0} (\beta - n_0)}$$

$$= \frac{\lim_{\beta \rightarrow n_0} r_2(\alpha, \beta)}{\lim_{\beta \rightarrow n_0} 1}$$

(by L'Hospital's rule)

$$= r_2(\alpha, n_0). \quad (5.66)$$

The various forms for $G(\alpha, \beta, n_0)$, in terms of $G^*(\alpha, \beta)$ are listed in Table 5.8.

Note that the special case $n_0=0$, $\beta=0$ and $\tilde{\tau}_1=0$ reduces to either $r_2(\alpha, n_0)$ or $r_1(\alpha)$ depending on the class of model. The convention of assuming that $\beta=0$ identifies the no initial cracking class is made by GVAL. (By setting $\tilde{\tau}_1$ to be at least ϵ , where ϵ is a very small number, in PSI ensures that GVAL will never be called with $\beta=0$ for a cracked structure).

The loss factor evaluation is thus based on the use of a single function called GVAL which is described below. A value of the loss factor, $H(\alpha, \beta, n_0, t)$ is obtained via,

$$H(\alpha, \beta, n_0, t) = \text{EXP1}(-\text{RNSV} * \text{GVAL}(\text{ALPV}, \text{BETV})). \quad (5.67)$$

	$\beta > n_0$	$\beta = n_0$
$n_0 > 0$	$\frac{G^*(\alpha, \beta) - G^*(\alpha, n_0)}{(\beta - n_0)}$	$r_2(\alpha \psi(n_0))$
$n_0 = 0$	$G^*(\alpha, \beta) / \beta$	$r_1(\alpha \psi(n_0))$

Table 5.8. Various forms of $G(\alpha, \beta, n_0)$ in terms of $G^*(\alpha, \beta)$ for the different combinations of values of β and n_0 .

The exception to this is when an evaluation of $1-H(\alpha, \beta, n_0, t)$ is required and $GVAL(\alpha, \beta, n_0)$ is very small, ($< 10^{-4}$ say). In this case, the series approximation to the exponential function is used to yield

$$1-H(\alpha, \beta, n_0, t) = RNSV * GVAL(ALPV, BETV), \quad (5.68)$$

as used, for example, in FALP.

The function $G^*(\alpha, \beta)$ is interpolated in (α, β) space by tabulating values of $\log(G^*(\alpha, \beta))$ and, generally, using bilinear interpolation to yield $\log(G^*(\alpha, \beta))$ for intermediate values of α and β . Special interpolation procedures are required near some of the boundaries of the interpolation region.

The function $GVAL$ is described below. The structure of the interpolation table and the methods of initialisation and interpolation are described in subsequent sections.

FUNCTION GVAL(ALPV,BETV)

Function

GVAL returns the value of the function $G(\alpha, \beta, n_0)$ for given values of α, β and n_0 (which is set in COMMON). The function $G(\alpha, \beta, n_0)$ is used to evaluate the loss factor,

$$H(\alpha, \beta, n_0, t) = \exp\{-t \cdot G(\alpha, \beta, n_0)\} \quad (5.69)$$

Parameter List

ALPV: Value of α for which the evaluation is required.

BETV: Value of β for which the evaluation is required.

Operation

The function GVAL is primarily concerned with selecting the special cases according to Table 5.3. The code is a straightforward implementation of this Table.

The function GSTAR is used to obtain values of $G^*(\alpha, \beta)$ via interpolation.

5.4.1. Structure of the Interpolation Table

The limits of integration for the double integral terms in $p_F(t, n_0)$ and $p_{fs}(t, n_0)$ for the most general model (equations (3.93) and (3.96) respectively) define the maximum possible ranges for α and β . These limits are,

$$\beta_1(n_0) \leq \beta \leq \tilde{\tau}_f^*(R_{\min}) \quad (5.70)$$

$$\alpha^*(n_0, \beta) \leq \alpha \leq \alpha_2(\beta, R_{\max}). \quad (5.71)$$

Allowing for all possible values of n_0 and neglecting the inspection boundaries, the loss factor interpolation table is required to span a space defined by

$$\tilde{\tau}_1 \leq \beta \leq \tilde{\tau}_f \quad (5.72)$$

and

$$\max(\alpha_{\min}, R_{\min}/\psi(\beta)) \leq \alpha \leq \min(\alpha_{\max}, R_{\max}/\psi(\beta)) \quad (5.73)$$

The nodal values of α and β at which the values of $G^*(\alpha, \beta)$ are computed are arranged so that all the nodes fall within the confines of the region of (α, β) space, defined above, according to the following system.

The β values are set by the nodal values of $\psi(\beta)$, together with the values, $\psi^{-1}(R_{\max}/\alpha_{\max})$, $\psi^{-1}(R_{\min}/\alpha_{\min})$ and $\psi^{-1}(R_{\min}/R_{\max})$ if they lie within the interval, $(\tilde{\tau}_1, \tilde{\tau}_f)$. Along each β line, K_{lim} α values are distributed according to following strategy. For the given value of β , β_j say, α ranges from α_{1j} to α_{2j} where,

$$\alpha_{1j} = \max(\alpha_{\min}, R_{\min}/\psi(\beta_j)) \quad (5.74)$$

$$\alpha_{2j} = \min(\alpha_{\max}, R_{\max}/\psi(\beta_j)) \quad (5.75)$$

The function $G^*(\alpha, \beta)$ can be discontinuous across the line $\alpha = R_{\max}$. Because of this, this value of α is also included as a nodal value on each β line. Note that the default limits for $\tilde{\tau}_1$ and $\tilde{\tau}_f$, (equations (3.45) and (3.46)), ensure that $\alpha_{1j} \leq R_{\max} \leq \alpha_{2j}$, thereby defining two subintervals $[\alpha_{1j}, R_{\max}]$ and $[R_{\max}, \alpha_{2j}]$. In each subinterval, the α nodes are distributed uniformly with K_1+1 alpha values in the first subinterval and K_2+1 values in the second. The numbers K_1 and K_2 are set so that,

$$K_1 = (R_{\max} - \alpha'_{\min}) / (\alpha'_{\max} - \alpha'_{\min}) * (K_{lim} - 1) \quad (5.76)$$

$$\text{and } K_2 = K_{lim} - K_1 - 1. \quad (5.77)$$

$$\text{where } \alpha'_{\min} = \min_j \{\alpha_{1j}\} \quad (5.78)$$

$$\text{and } \alpha'_{\max} = \max_j \{\alpha_{2j}\} \quad (5.79)$$

Geometrically, this generates a non-orthogonal interpolation mesh of the form shown in Figure 5.11. The mesh satisfactorily follows boundary curves across which the function $G^*(\alpha, \beta)$ can be discontinuous. In the mesh shown in Figure 5.11 there are two points at which the mesh lines converge. Such points cause no difficulty during interpolation: the table simply contains several identical entries representing the same point but different mesh lines and since inverse interpolation is not required, the degeneracy can be easily handled. In general, there are three such points corresponding to the following conditions.

$$\alpha = R_{\max}, \quad \beta = \tilde{\tau}_1; \quad (5.80)$$

$$\alpha = R_{\max}, \quad \beta = \gamma^{-1}(R_{\min}/R_{\max}); \quad (5.81)$$

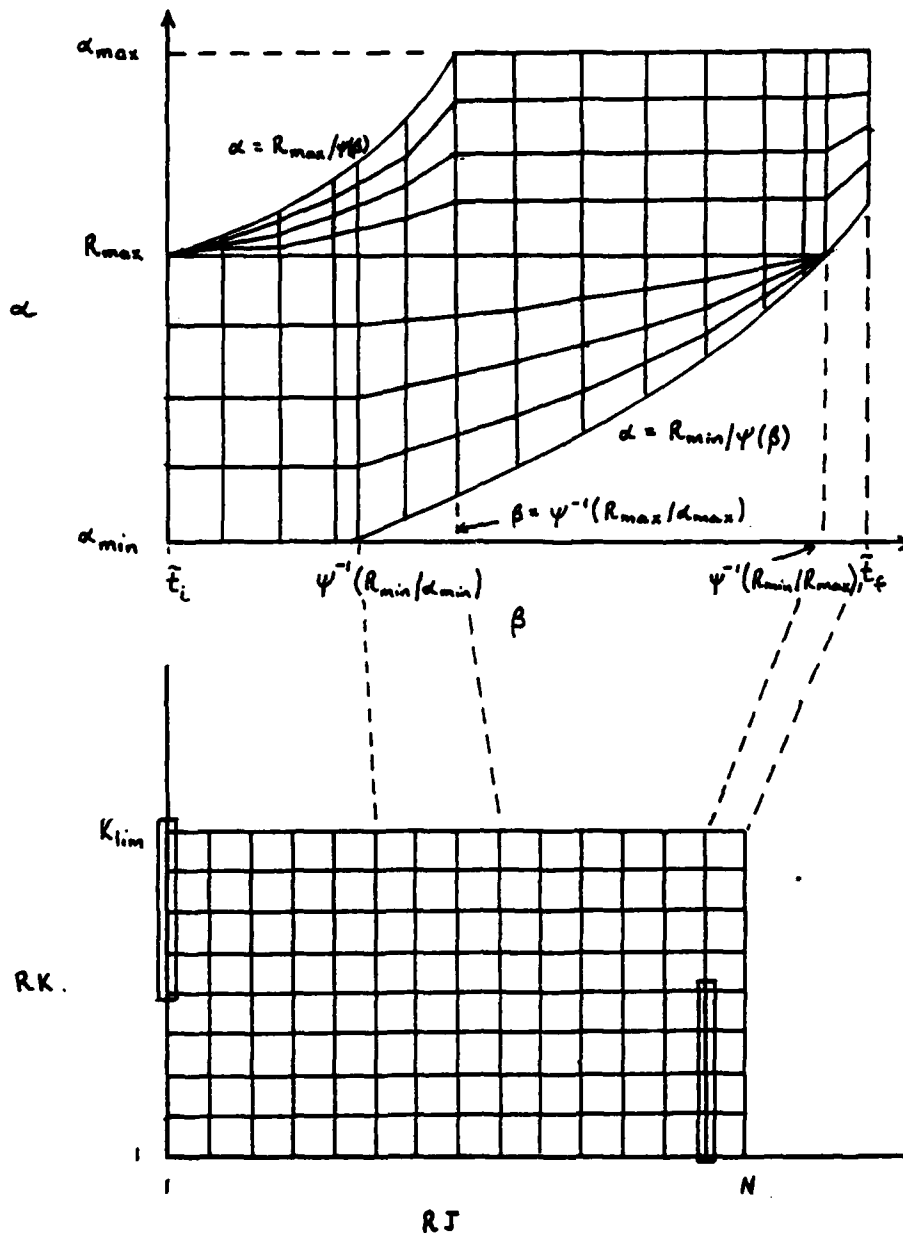


Figure 5.11 Construction of a typical loss factor interpolation mesh. The mesh is constructed using the N nodes in β used to define $\psi(\beta)$. The α nodes have been selected according to the strategy described in the text; for simplicity K_1 and K_2 have both been set to 4, with $K_{lim} = 9$.

$$\alpha = \alpha_{\max}, \quad \beta = \psi^{-1}(R_{\min}/\alpha_{\max}). \quad (5.82)$$

The definition of the mesh may be considered in terms of a mapping from the variables α, β to two mesh variables RK, RJ where RK and RJ have values in the intervals $[1, K_{1\max}]$ and $[1, N]$ respectively. (N is the number of nodes in the sequence of ordered pairs defining $\psi(\beta)$, including the three special values defined above and inserted by SETTAB.) In the mesh defined above, lines of constant RJ correspond to lines of constant β . If β lies in the interval $[\beta_j, \beta_{j+1})$, RJ is given by

$$RJ = (\beta - \beta_j)/(\beta_{j+1} - \beta_j). \quad (5.83)$$

Conversely, β is given by

$$\beta = \beta_j + (\beta_{j+1} - \beta_j) * (RJ - j). \quad (5.84)$$

The relationship between α and RK is a little more complex. Given K_1 and K_2 , it is clear that for all β , the line $RK = K_1 + 1$ corresponds to the line

$$\alpha = \alpha_{\text{sep}} = \max\{\alpha_{\min}, \min\{R_{\max}, \alpha_{\max}\}\}. \quad (5.85)$$

Hence, given (α, β) , RK can be computed via the following steps.

$$(1) \text{ Compute } \alpha_{\text{low}} = \max\{\alpha_{\min}, R_{\min}/\psi(\beta)\} \quad (5.86)$$

$$\text{and } \alpha_{\text{high}} = \min\{\alpha_{\max}, R_{\max}/\psi(\beta)\}. \quad (5.87)$$

(11) If $\alpha > \alpha_{\text{sep}}$;

$$RK = K_1 + 1 + (\alpha - \alpha'_{\text{low}})/(\alpha_{\text{high}} - \alpha'_{\text{low}}) * K_2 \quad (5.88)$$

$$\text{where } \alpha'_{\text{low}} = \max(\alpha_{\text{sep}}, \alpha_{\text{low}}). \quad (5.89)$$

(iii) If $\alpha \leq R_{\text{max}}$;

$$RK = 1 + (\alpha - \alpha_{\text{low}}) / (\alpha_{\text{sep}} - \alpha_{\text{low}}) * K_1. \quad (5.90)$$

In (ii), if $|\alpha_{\text{high}} - \alpha_{\text{sep}}| < \epsilon$, RK is arbitrarily set to K_1+1 .

In (iii), if $|\alpha_{\text{sep}} - \alpha_{\text{low}}| < \epsilon$, RK is arbitrarily set to K_1+1 .

Conversely, given RK, α can be computed via the following steps.

(i) Compute α_{low} , α_{high} as in (i) above.

(ii) If $RK > K_1+1$;

$$\alpha = \alpha'_{\text{low}} + (\alpha_{\text{high}} - \alpha'_{\text{low}}) \cdot (RK - K_1 - 1) / K_2 + \alpha'_{\text{low}} \quad (5.91)$$

(iii) If $RK \leq K_1+1$;

$$\alpha = \alpha_{\text{low}} + (\alpha_{\text{sep}} - \alpha_{\text{low}}) \cdot (RK - 1) / K_1 + \alpha_{\text{low}}. \quad (5.92)$$

The above equations completely specify the transformation. Note that constant RK lines do not necessarily map into straight lines in α, β space, but follow curves dictated by $\varphi(\beta)$.

A typical interpolation mesh generated by this logic is shown in Figure 5.12.

355

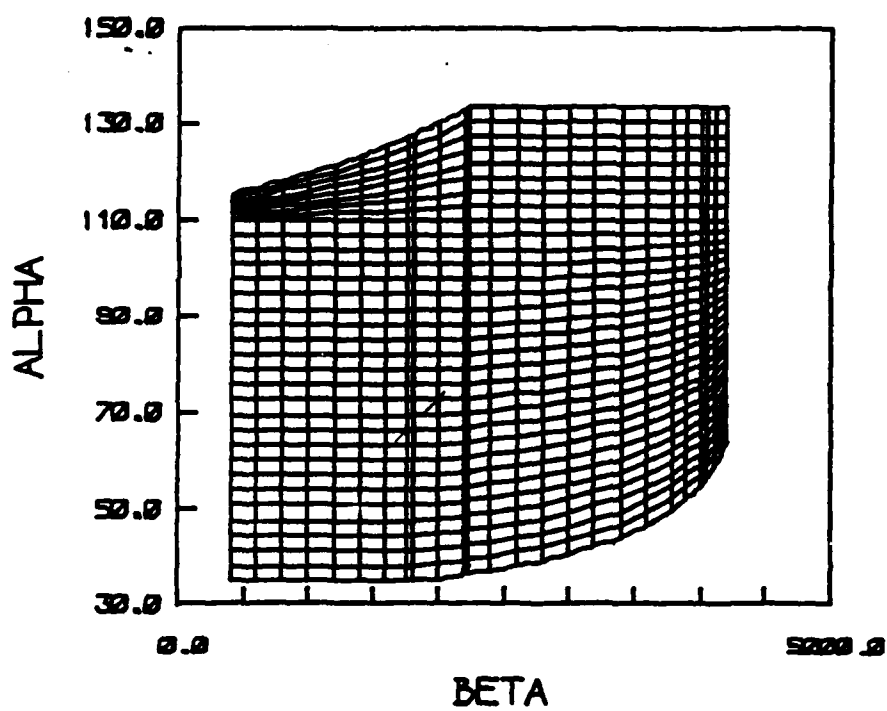


Figure 5.12. Typical loss factor interpolation mesh.
(Example A.) ($K_{lim} = 33$)

FUNCTION ALPTAB(RK,ALPLOW,ALPHIG)

Function

ALPTAB returns a value of α corresponding to RK for the β line denoted by ALPLOW, ALPHIG where

$$\text{ALPLOW} = \max\{\alpha_{\min}, R_{\min}/\psi(\beta)\} \quad (5.93)$$

$$\text{ALPHIG} = \min\{\alpha_{\max}, R_{\max}/\psi(\beta)\}. \quad (5.94)$$

Parameter List

RK: Value of RK for which the α value is required.

ALPLOW: Lower limit for α on the β line.

ALPHIG: Upper limit for α on the β line.

Operation

Broadly, ALPTAB executes the transformation defined by equations (5.91) and (5.92) where (5.86) and (5.87) have been set by the calling code to produce ALPLOW and ALPHIG.

Using the notation of Section 5.4.1, ALPTAB assumes that the following constants have been set.

$$\text{RK1} = K_1 + 1 ; \quad K1 = K_1 + 1 \quad (5.95)$$

$$\text{ALPSEP} = \text{AMAX1}(\text{ALPL}, \text{AMIN1}(\text{ALPH}, R(M))) \quad (5.96)$$

where

$$\text{ALPL} = \alpha'_{\min} \quad (5.97)$$

and

$$\text{ALPH} = \alpha'_{\max}. \quad (5.98)$$

$$RKIM1 = K_1 \quad (5.99)$$

$$RKLM1 = K_2. \quad (5.100)$$

Then if $RK < K1$, α is given by equation (5.92), i.e.,

$$ALPTAB = ALPLOW + (ALPSEP - ALPLOW) * (RK - 1.0) / RKLM1. \quad (5.101)$$

If $RK > K1$, α is given by equation (5.91), i.e.,

$$ALPTAB = ALPL2 + (ALPHIG - ALPL2) * (RK - RK1) / RKLM1 \quad (5.102)$$

where

$$ALPL2 = \max\{ALPLOW, ALPSEP\}. \quad (5.103)$$

FUNCTION INITAB(DUM)

Function

Function INITAB initialises the constants required by functions ALPTAB and RKTAB. No function value is returned.

Parameter List

DUM: Dummy parameter. (The dummy parameter is required because INITAB is an entry point in the function ALPTAB. In an installation where entry points are not permitted INITAB could be rewritten as a subrotuine.)

Operation

Given the constant KLIM (K_{lim}), the number of K nodes, the following operations are performed by INITAB.

$$(i) \text{ Compute } \alpha'_{min} = \min_j (\alpha_1(j)) = ALPL. \quad (5.104)$$

$$(ii) \text{ Compute } \alpha'_{max} = \max_j (\alpha_2(j)) = ALPH. \quad (5.105)$$

$$(iii) \text{ Compute } \alpha'_{int} = (\alpha'_{max} - \alpha'_{min}) / (K_{lim} - 1) \quad (5.106)$$

$$(iv) \text{ Set } ALPSEP = \alpha_{sep} = R_{max}. \quad (5.107)$$

(This value may be overwritten below.)

(v) Calculate K_1 ;

$$\text{If } \alpha_{sep} \leq \alpha_{min}; K_1=1, (K_1=0). \quad (5.108)$$

$$\text{If } \alpha_{sep} \geq \alpha_{max}; K_1=KLIM+1, (K_1=K_{lim}). \quad (5.109)$$

Otherwise, in accordance with equation (5.76),

$$\begin{aligned}
 K1 &= \text{integ}((\alpha_{\text{sep}} - \alpha_{\text{low}})/\alpha_{\text{int}}) + 1, \\
 &= \text{IFIX}((\text{ALPSEP} - \text{ALPL})/\text{ALPINT} + 1. \quad (5.110)
 \end{aligned}$$

(vi) Set the following,

$$\text{RK1} = K1 = K_1 + 1 \quad (5.111)$$

$$\text{ALPSEP} = \alpha_{\text{sep}} = \max(\alpha_{\text{low}}, \min(\alpha_{\text{high}}, R_{\text{max}})) \quad (5.112)$$

$$\text{RK1M1} = K_1 = K1 - 1 \quad (5.113)$$

$$\text{RKLM1} = K_2 = \text{RKLM} - K1 \quad (5.114)$$

$$\text{TEST} = \text{EPS2} * \text{ALPSEP} \quad (5.115)$$

where EPS2 is a small number close to machine accuracy. (Test is used to trap cases where $\alpha = \alpha_{\text{sep}}$ in RKTAB.) If $K_1 = K_{\text{lim}} + 1$, $K_2 = 0$ and K_1 is incremented by 1 to prevent ALPTAB from entering the section of code associated with the α nodes having α greater than α_{sep} .

FUNCTION RKTAB(ALPVAL,ALPLOW,ALPHIG)

Function

RKTAB returns the value of RK corresponding to α for the line denoted by ALPLOW, ALPHIG where

$$\text{ALPLOW} = \max(\alpha_{\min}, R_{\min}/\psi(\beta)), \quad (5.116)$$

$$\text{ALPHIG} = \min(\alpha_{\max}, R_{\max}/\psi(\beta)). \quad (5.117)$$

Parameter List

ALPVAL: Value of α for which RK is required.

ALPLOW: Minimum value of α for the β line.

ALPHIG: maximum value of α for the β line.

Operation

Broadly, RKTAB executes the sequence of operations described by equations (5.88) to (5.90).

Using the notation of Section 5.4.1, RKTAB assumes that the following constants have been initialised.

$$\text{RK1} = K_1 + 1, \quad (5.118)$$

$$\text{ALPSEP} = \max(\alpha_{\text{low}}, \min(\alpha_{\text{high}}, R_{\text{max}})) \quad (5.119)$$

$$\text{RK1M1} = K_1 \quad (5.120)$$

$$\text{RKLM1} = K_2. \quad (5.121)$$

Then if $\alpha < \alpha_{\text{sep}}$, RK is given by equation (5.88),

$$\text{RK} = 1 + (\text{ALPVAL} - \text{ALPLOW})/(\text{ALPSEP} - \text{ALPLOW}) * \text{RK1M1}. \quad (5.122)$$

If $\alpha \geq \alpha_{\text{sep}}$, RK is given by equation (5.84),

$$RK = RK1 + (ALPVAI - ALPL2)/(ALPHIG - ALPL2)*RKLM1, \quad (5.123)$$

where

$$ALPL2 = \text{AMAX1}(ALPSEP, ALFLOW) = \alpha'_{\text{low}}. \quad (5.124)$$

If the denominator in either expression is nearly zero, RK = RK1 is returned.

5.4.2. Initialisation

The loss factor interpolation table is initialised by subroutine SETTAB which was partially documented in Section 5.2.1. This description of the initialisation of the loss factor completes the documentation for that subroutine.

The first operation associated with the loss factor interpolation table is the calculation of the α limit arrays which contain the limits α_{1j} and α_{2j} (equations (5.74) and (5.75) respectively) for each value of β_j in the sequence of ordered pairs defining $\psi(\beta)$.

Following the calculation of these arrays, the number of intervals in α , K_{lim} , is set according to the space available in the G array in COMMON block GCOM. Currently, K_{lim} is set to the minimum of 33 or $2500/N$ where N is the number of β_j values.

The table is initialised by the nested DO LOOPS terminating in statements 140 and 150. They sweep through the values of β_j and α values corresponding to the mesh nodes of the interpolation grid described in Section 5.4.1. (Note the use of ALPTAB to effect the mapping from integer values of K to α .)

The table, as stored on disk, contains values of the term,

$$\log \left(\int_{\alpha}^{\beta} r_2(\alpha\psi(\beta')) d\beta' / l_r \right)$$

so that the table is independent of the parameter l_r and the state of the VIRGIN switch. Accordingly, the VIRGIN switch is cancelled and value returned by RINTV divided by l_r . On completion of the table initialisation the VIRGIN switch is restored to its proper value. The logarithm is taken by the function LOGI which ensures that a zero argument value results in a known value of -80.0 for the logarithm. This fact is used by subsequent code to detect mesh points where the integral term is zero.

Once these table values have been calculated or read from a disk file, they are updated to include the factor 1_r and the term $r_1(\alpha)\tilde{t}_1$, if required. This is done by the DO LOOPS terminating at statements 250 and 260.

SETTAB provides for the construction of contour maps of the 'G integral' as written to disk or the 'G function' ($G(\alpha, \beta, n_0)$). The latter is computed in an obvious way by the DO LOOPS terminating at statements 280 and 310.

The subroutine NODES and the functions RINTV and RLGAM which are used to evaluate the table entries are described below.

SUBROUTINE NODES (GAMMA, IK)

Function

NODES determines the limits of integration and the nodal values of β for the integration term,

$$\text{Term} = \int_{\tilde{\tau}_1}^{\beta} r_2(\alpha \psi(\beta')) d\beta' . \quad (5.125)$$

Parameter List

GAMMA: Array into which the nodal values of β are returned to the calling code. (The limits of integration are returned as GAMMA(1) and GAMMA(IK).)

IK: On entry, the number of storage locations available in GAMMA.

On exit, the number of nodes placed in GAMMA. If IK=0, the integral term is zero.

Operation

The nodes for β correspond with the β values used to define $\psi(\beta)$ and the equivalents of the R values used to define $r_2(R)$. The nodal values for $\psi(\beta)$ are stored in the array BETA in the COMMON block PSICOM. The second set is obtained from the R nodes stored in the array R in the COMMON block LOADCM via the relationship,

$$\begin{aligned} \beta &= \psi^{-1}(R/\alpha) \\ &= \text{PSINV}(R(I)/\text{ALPV}). \end{aligned} \quad (5.126)$$

The limits of integration are defined by,

$$\max\{\tilde{\tau}_1, \psi^{-1}(R_{\max}/\alpha)\} \leq \beta' \leq \min\{\beta, \psi^{-1}(R_{\min}/\alpha)\}. \quad (5.127)$$

The required sequence of β values is produced by merging these

two sequences subject to the vignetting specified by equation (5.117), using the subroutine MERGE.

Note that if $\alpha\psi(\beta) \geq R_{\max}$, the integral term is zero and a value of 0 is returned for IK.

COMMON Variables Changed

ALPV: Current value of α .

BETV: Current value of β .

FUNCTION RINTV(ALPV1,BETV1)

Function

RINTV returns the value of the term,

$$\text{RINTV} = \int_{\xi_i}^{\beta} r_2(\alpha \psi(\beta')) d\beta' + [r_1(\alpha) \tilde{\epsilon}_1] \quad (5.128)$$

for given values of α and β . The term in square brackets is included only if the VIRGIN switch is on.

Parameter List

ALPV1: Value of α for which the evaluation is required.

BETV1: Value of β for which the evaluation is required.

Operation

After setting the current values of ALPV and BETV in the COMMON block PARCOM, subroutine NODES is called to ascertain the limits of integration and the nodal values of β' to which the integration procedure is to be locked. (See NODES for details.). The adaptive integration routine ADAPT2 is used to evaluate the integral term.

Note that this function can be used, if required to replace the interpolation procedure for evaluating the function $G^*(\alpha, \beta)$.

Internal storage allocated for the GAMMA array is currently set within RINTV to 150.

FUNCTION RLGAM(ARG)

Function

RLGAM returns the value of $r_2(\alpha\psi(\beta))$ given a values of (ARG) and α (ALPV in COMMON).

Parameter List

ARG: Value of β for which $r_2(\alpha\psi(\beta))$ is required.

Operation

RLGAM uses the value of ALPV in COMMON block PARCOM to calculate $ARG1 = \alpha\psi(\beta)$ which is used, in turn, as the argument of RLOAD to return the required value of $r_2(\alpha\psi(\beta))$.

5.4.3. Interpolation

The loss factor table contains values of $\log(G^*(\alpha, \beta))$ for values of α and β distributed throughout (α, β) space according to the scheme described in Section 5.4.1. Denoting the α value corresponding to $RK=k$ on the line $\beta = \beta_j$ by α_k , the table entries can be denoted by $LG_{j,k}$ where

$$LG_{j,k} = \log(G^*(\alpha_k, \beta_j)). \quad (5.129)$$

The function $G^*(\alpha, \beta)$ can vary rapidly with α and β and direct polynomial interpolation was found to be inadequate. In particular, small oscillations in the interpolated function, although not significant in terms of the overall accuracy of the reliability functions, were found to require excessive effort by the adaptive integration routines to achieve convergence. The procedure finally adopted is based on the use of bilinear interpolation for the function $\log(G^*(\alpha, \beta))$. Interpolation is made in the α direction first. (Note that the mesh is non-orthogonal: the sequence of interpolation can affect the final interpolated value.)

For given values of α and β such that $k \leq RK < k+1$ and $\beta_j \leq \beta < \beta_{j+1}$ where RK corresponds to α for the line $\beta = \beta$, the interpolation in the α direction yields two values, LG_1 and LG_2 , of $\log(G^*(\alpha, \beta))$ at the end points of the line having the given value of RK , (see Figure 5.13). The relevant equations are,

$$LG_1 = LG_{j,k} + (LG_{j,k+1} - LG_{j,k})(RK-k) \quad (5.130)$$

and

$$LG_2 = LG_{j+1,k} + (LG_{j+1,k+1} - LG_{j+1,k})(RK-k). \quad (5.131)$$

Interpolation in the β direction yields,

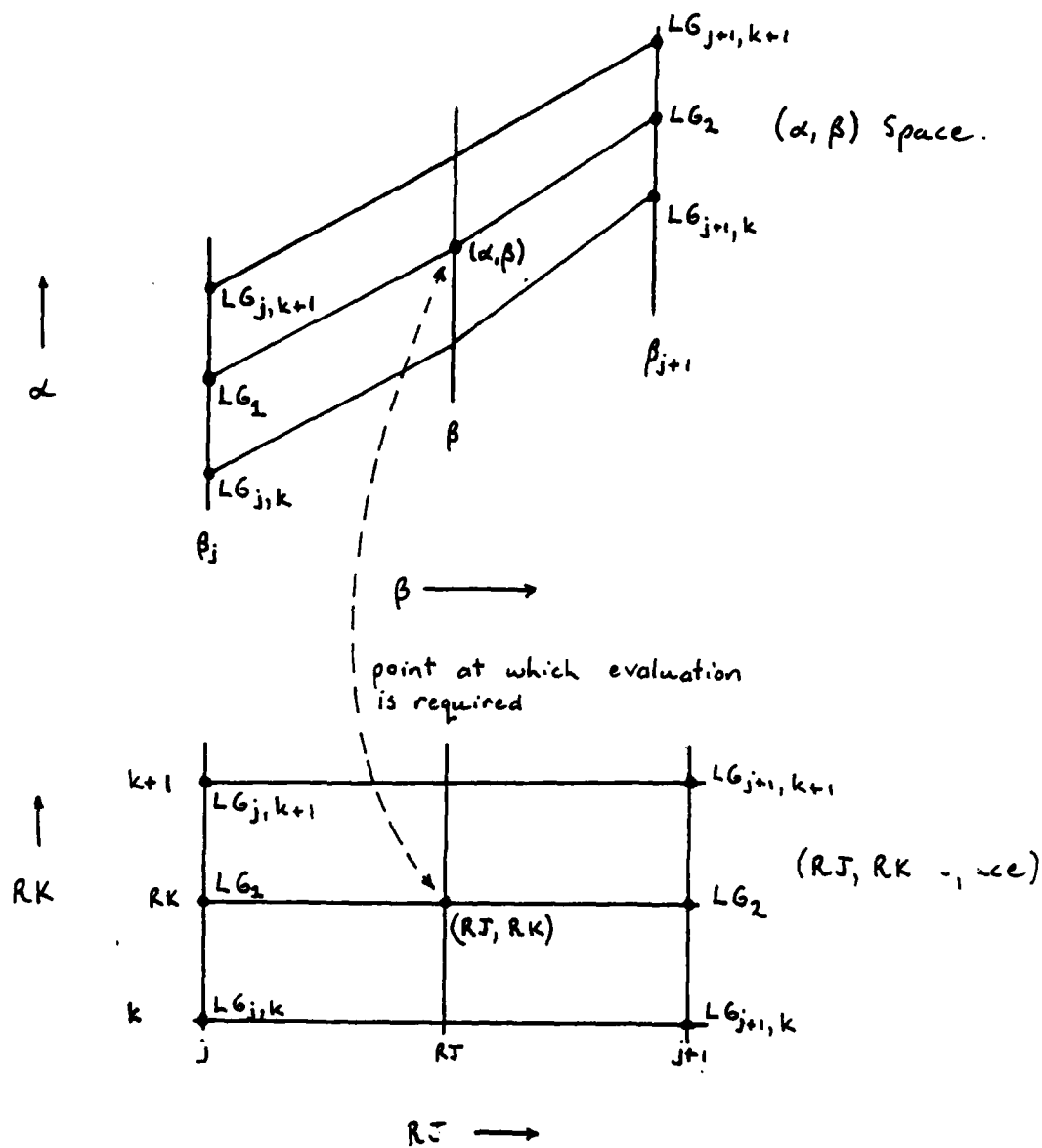


Figure 5.13. Notation used during interpolation of the loss factor table, (see equations (5.130) and (5.131)).

$$G^*(\alpha, \beta) = \exp(LG_1 + (LG_2 - LG_1)(\beta - \beta_j)/(\beta_{j+1} - \beta_j)). \quad (3.132)$$

The use of bilinear logarithmic interpolation does, of course, require careful consideration in regions surrounding points where $G^*(\alpha, \beta) = 0$. Fortunately, the form of the function $G^*(\alpha, \beta)$ (equation (5.63)) is such that these points are confined to easily identified boundary locations and special interpolation procedures can be devised and incorporated into the overall scheme. The conditions leading to $G^*(\alpha, \beta) = 0$ are,

$$\alpha\psi(\beta) = R_{\max} \quad (5.133)$$

$$\beta = \tilde{\epsilon}_1 ; \text{ for VIRGIN} = \text{false} \quad (5.134)$$

and

$$\beta = \epsilon \text{ for } \tilde{\epsilon}_1 = \epsilon, \quad (5.135)$$

where ϵ is the small number used by PSI to set a minimum value for $\tilde{\epsilon}_1$.

The procedures associated with each of these special boundaries are described below.

$$(1) \underline{\alpha\psi(\beta) = R_{\max}}$$

The condition $\alpha\psi(\beta) = R_{\max}$ corresponds to $\alpha_{2j} = R_{\max}/\psi(\beta)$ and applies only to α values having $RK = K_{lim}$. It is appropriate, therefore, to consider the treatment of this special condition when interpolating in the direction along the nodal β lines ($\beta = \beta_j$ say). A special interpolation formula is required for $\alpha_{K_{lim}-1} < \alpha < \alpha_{K_{lim}}$ and $G^*(\alpha_{K_{lim}}, \beta_j) = 0$.

From the structure of the interpolation table, if $\alpha_{K_{lim}}\psi(\beta_j) = R_{\max}$, then $\alpha_{K_{lim}-1} < R_{\max}$. Consequently, the points of intersection of the lines $\alpha = \alpha$ and $\alpha = \alpha_{K_{lim}-1}$

with $\alpha\psi(\beta)=R_{\max}$, yield values of β , β' and $\beta'_{K_{lim}-1}$ respectively, where

$$\beta' = \psi^{-1}(R_{\max}/\alpha) \quad (5.136)$$

and

$$\beta'_{K_{lim}-1} = \psi^{-1}(R_{\max}/\alpha_{K_{lim}-1}) \quad (5.137)$$

such that β' and $\beta'_{K_{lim}-1}$ are both greater than $\tilde{\epsilon}_1$, (as shown in Figure 5.14). The function $G^*(\alpha, \beta)$ is therefore given by

$$G^*(\alpha, \beta) = \int_{\beta'}^{\beta_j} r_2(\alpha, \beta') d\beta' \quad (5.138)$$

which can be approximated by

$$G^*(\alpha, \beta) \approx C \cdot r_2(\alpha\psi(\beta_j)) \cdot (\beta_j - \beta'). \quad (5.139)$$

On $\beta = \beta_j$, $r_2(\alpha, \beta)$ can be represented approximately by

$$\begin{aligned} \log(r_2(\alpha, \beta_j)) &\approx \log(r_2(\alpha_{K_{lim}}, \beta_j)) \\ &\quad + (\alpha - \alpha_{K_{lim}-1}) / (\alpha_{K_{lim}} - \alpha_{K_{lim}-1}) \\ &\quad * \log(r_2(R_{\max}) / r_2(\alpha_{K_{lim}-1}, \beta_j)). \end{aligned} \quad (5.140)$$

Equations (5.139) and (5.140) can be combined to yield

$$\begin{aligned} \log(G^*(\alpha, \beta_j)) &= \log(C) + \log(\beta_j - \beta') \\ &\quad + \log(r_2(\alpha_{K_{lim}-1}, \beta_j)) + (\alpha - \alpha_{K_{lim}-1}) / (\alpha_{K_{lim}} - \alpha_{K_{lim}-1}) \\ &\quad * \log(r_2(R_{\max}) / r_2(\alpha_{K_{lim}-1}, \beta_j)). \\ &\quad \dots \end{aligned} \quad (5.141)$$

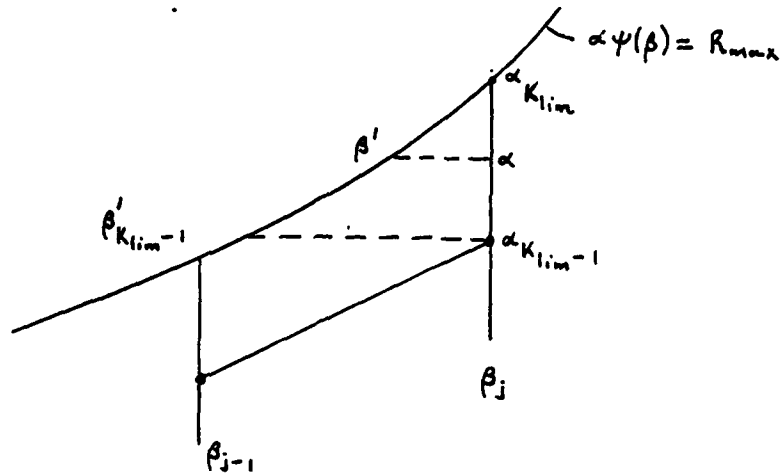


Figure 5.14. Notation used to interpolate in the α direction on the line $\beta = \beta_j$ in the vicinity of the line $R_{\max} = \alpha\psi(\beta)$.

By enforcing continuity at $(\alpha_{K_{lim}-1}, \beta_j)$,

$$\begin{aligned} \log(C) &= \log(G^*(\alpha_{K_{lim}-1}, \beta_j)) \\ &- \log(\beta_j - \beta_{K_{lim}-1}) - \log(r_2(\alpha_{K_{lim}-1}, \beta_j)), \quad (5.142) \end{aligned}$$

so that

$$\begin{aligned} \log(G^*(\alpha, \beta_j)) &= LG_{K_{lim}-1, j} + \log((\beta_j - \beta')/(\beta_j - \beta_{K_{lim}-1})) \\ &+ (\alpha - \alpha_{K_{lim}-1})/(\alpha_{K_{lim}} - \alpha_{K_{lim}-1}) \\ &\quad * \log(r_2(R_{max})/r_2(\alpha_{K_{lim}-1}, \beta_j)) \\ &\dots\dots \quad (5.143) \end{aligned}$$

(11) $\beta = \tilde{\tau}_1$ for VIRGIN = false

The case of $G^*(\alpha, \beta) = 0$ at $\beta = \tilde{\tau}_1$ can be considered as a special case of interpolation in the β direction. Given β_1 and β_2 , the end points of the RK line corresponding to (α, β) can be denoted by (α_1, β_1) and (α_2, β_2) . The corresponding values of $\log(G^*(\alpha, \beta))$ are LG_1 and LG_2 . This special case corresponds to an infinite value for LG_1 . (In practice the logarithm function ALOG1 will return a value of -80.0 for a zero argument.) Note that that generally $\psi(\beta) \approx 1$ in the vicinity of $\beta = \tilde{\tau}_1$ so that over the first interval in β , $\alpha_1 \approx \alpha_2$.

The special interpolation function for this case can be found by generating a linear interpolation formula for $\log(G^*(\alpha, \beta) + C)$ where C is a constant which is found by looking closely at the form of $G^*(\alpha, \beta)$ as $\beta \rightarrow \tilde{\tau}_1$. The linear interpolation equations leads to

$$\begin{aligned} G^*(\alpha, \beta) &= \exp(\log(C) + (\beta - \beta_1)/(\beta_1 - \beta_2) \\ &\quad * \log((\exp(LG_2) + C)/C)) - C. \quad (5.144) \end{aligned}$$

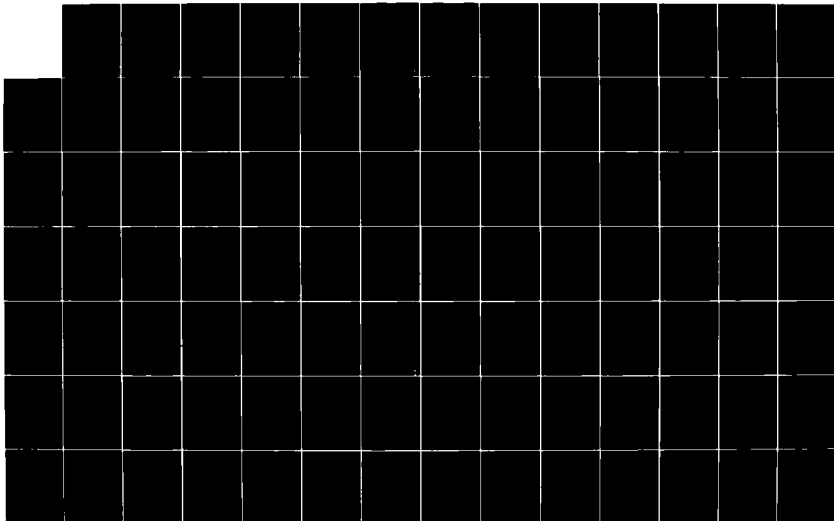
AD A145 685

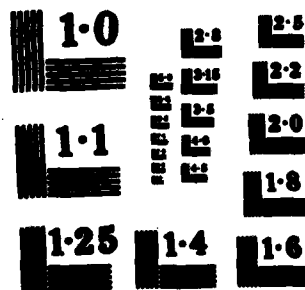
NERT A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUN. (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARL/STRUC 397

5/7

1/6 9/2

NI





Noting that

$$\frac{\partial G^*}{\partial \beta}(\alpha, \beta) = r_2(\alpha, \psi(\beta)),$$

$$r_2(\alpha, \psi(\beta)) = (G^*(\alpha, \beta) + C) \log((G^*(\alpha_2, \beta_2) + C)/C) / (\beta_2 - \beta_1) \quad (5.145)$$

.....

so that,

$$r_2(\alpha_2, \beta_2) = (G^*(\alpha_2, \beta_2) + C) \log((G^*(\alpha_2, \beta_2) + C)/C) / (\beta_2 - \beta_1)$$

and

$$r_2(\alpha_1, \beta_1) = C \cdot \log((G^*(\alpha_2, \beta_2) + C)/C) / (\beta_2 - \beta_1)$$

and finally

$$C = G^*(\alpha_2, \beta_2) r_2(\alpha_1, \beta_1) / (r_2(\alpha_2, \beta_2) - r_2(\alpha_1, \beta_1)) \quad (5.146)$$

.....

The special interpolation formula is given by equation (5.145) with C defined by (5.146).

If $r_2(\alpha_1, \beta_1) \simeq r_2(\alpha_2, \beta_2)$, then $G(\alpha, \beta, 0)$ is constant and $G^*(\alpha, \beta)$ can be interpolated via,

$$G^*(\alpha, \beta) = \exp(LG_2) * (\beta - \beta_1) / (\beta_2 - \beta_1). \quad (5.147)$$

(iii) $\beta \rightarrow 0$

If $\beta_1 \simeq 0$, $G^*(\alpha, \beta) \simeq 0$. An appropriate formula for $G^*(\alpha, \beta)$ over the first β interval may be generated by assuming that $G(\alpha, \beta, 0)$ varies linearly from β_1 to β_2 . From equation (5.66),

$$\lim_{\beta \rightarrow 0} G(\alpha, \beta, 0) = r_2(\alpha, \beta) \quad (5.148)$$

375

and by definition,

$$G(\alpha, \beta_2, 0) = G^*(\alpha, \beta_2) / \beta_2. \quad (5.149)$$

The assumption that $G(\alpha, \beta, 0)$ varies linearly over the first interval leads to,

$$G^*(\alpha, \beta) = r_2(\alpha, \beta_1) \cdot \beta + (\exp(LG_2) - \beta_2 \cdot r_2(\alpha, \beta_1)) \cdot \beta^2 / \beta_2^2. \quad (5.150)$$

.....

The interpolated value of $G^*(\alpha, \beta)$ is provided by the subroutine GSTAR which uses the function GALP to obtain the terms LG_1 and LG_2 representing interpolations in the α direction.

FUNCTION GALP(J,RK)

Function

GALP returns the value of $\log(G^*(\alpha, \beta))$ corresponding to a given nodal value of β , β_j , and a given value of the α index, RK. This value of $G^*(\alpha, \beta)$ is obtained via interpolation in the α direction.

Parameter List

J: Value of j (corresponding to β_j) for which the evaluation is required. Note that GALP can provide interpolation along β_j lines only.

RK: Value of RK for which the evaluation is required.

Operation

The default interpolation procedure is a simple bilinear procedure using equation (5.130), i.e.,

$$\begin{aligned} \log(G^*(\alpha, \beta)) &= LG_{j,k} + (LG_{j,k+1} - LG_{j,k}) * (RK - k) \\ &= G(JK) + (G(JK+N) - G(JK)) * ALPFAC. \end{aligned} \quad (5.151)$$

If $JK = K_{lim}$, then the function GALP returns the value of $LG_{j, K_{lim}}$.

If $LG_{j,k+1} \leq -80.0$, then $G^*(\alpha_{k+1}, \beta_j)$ is zero and the special interpolation procedure identified in (1) of Section 5.4.3. is invoked. This interpolation formula (equation 5.141) is evaluated by the code following statement number 40.

If $LG_{j,k+1} \leq -80.0$ and $LG_{j,k} \leq -80.0$, a value of -80.0 is returned to indicate to the calling code that $G^*(\alpha, \beta)$ is zero.

Significant Local Variables

K: Largest integer less than RK, ($=k$).

G: Two dimensional array containing the values of $LG_{j,k}$.

JK: Index variable for G. $JK=j+(k-1)*N$ where N is the number of j lines in the interpolation table.

ALPFAC: $RK-K$.

B0: β_j , the value of the current β line.

BDIFF: $\beta_j - \beta'$.

FUNCTION GSTAR(ALPV,BETV)

Function

GSTAR evaluates the function $G^*(\alpha, \beta)$ for given values of α and β .

Parameter List

ALPV: Value of α for which the value of $G^*(\alpha, \beta)$ is required.

BETV: Value of β for which the value of $G^*(\alpha, \beta)$ is required.

Operation

The operation of GSTAR follows the logic shown in Figure 5.15.

Given α and β , the first operation associated with the evaluation of $G^*(\alpha, \beta)$ is the examination of the location of the point (α, β) in relation to the region spanned by the interpolation table. The following actions can be taken.

- (i) If $\alpha\psi(\beta) > R_{\max}$, $G^*(\alpha, \beta) = 0$ and no further action is taken by GSTAR.
- (ii) If α is such that $RK < 1$, $G^*(\alpha, \beta)$ is set to the value corresponding to $RK = 1$.
- (iii) If $RK > K_{11m} + \epsilon$, then the evaluation is considered to be illegal and an appropriate error message is issued. If $K_{11m} < RK < K_{11m} + \epsilon$, GSTAR returns the value corresponding to $RK = K_{11m}$.
- (iv) If $\beta > \beta_N$, β is out of range and an error message is issued.

GSTAR controls the interpolation in the β direction and relies on GALP to provide the values of LG_1 and LG_2 which result from interpolations in the α direction. The logic within GSTAR is associated with identifying the special conditions and

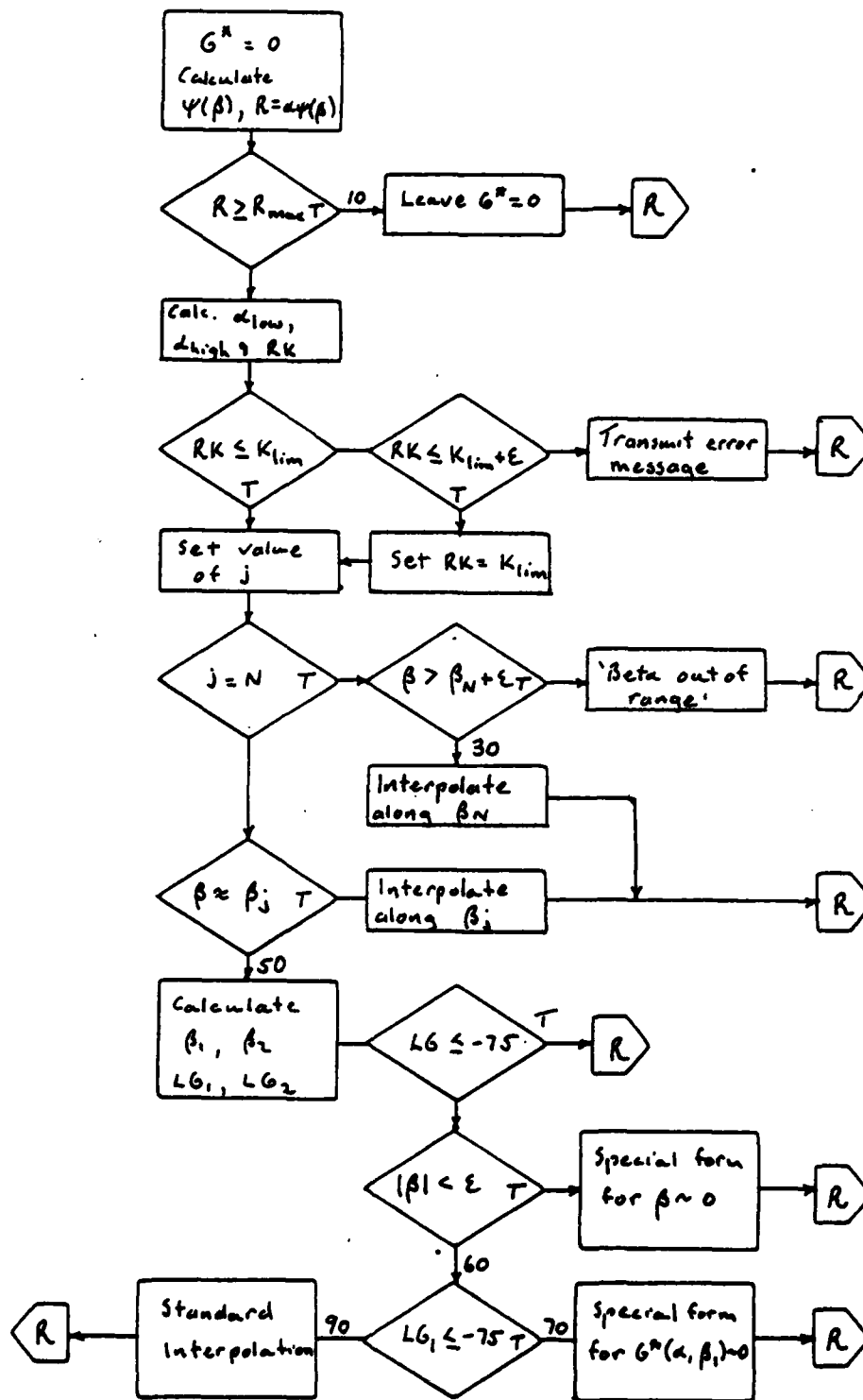


Figure 5.15. Logic used by the function GSTAR to evaluate $G^*(\alpha, \beta)$.

determining the appropriate interpolation formula as detailed below.

- (i) If $\beta \approx \beta_j$, no interpolation in β is required. A single call to GALP is used to calculate $G^*(\alpha, \beta)$.
- (ii) If $LG_2 \leq -80.0$, ($\exp(LG_2) = 0$), then GSTAR returns the value 0 for $G^*(\alpha, \beta)$.
- (iii) If $LG_1 \leq -80.0$, then the special case (ii) in Section 5.4.3. is identified. Equations (5.144) and (5.146) define the interpolating function. Note that this case can occur only for $j=1$.

If, for this special case, $r_2(\alpha_1, \beta_1) \approx r_2(\alpha_2, \beta_2)$, $G^*(\alpha, \beta)$ is linear over the β interval and (5.147) is used.

- (iv) If $\beta_1 \approx 0$, special case (iii) in Section 5.4.3 is identified and equation (5.150) is used.

The coding associated with each special case and the default interpolating function (5.132) is a straightforward coding of the relevant equations using the various local variables defined below.

Significant Local Variables

- PSIV: $\psi(\beta)$.
- RVAL: $\alpha\psi(\beta)$.
- ALPV1: α_{low} (equation 5.86). Also used to store α_1 for (5.146).
- ALPV2: α_{high} (equation 5.87). Also used to store α_2 for (5.147).
- RK: α index for interpolation table. (See Section 5.4.2.)

B1: β_j .

B2: β_{j+1} .

G1: LG_1 , equation (5.129). Also used to store $\log(C)$ for (5.144).

G2: LG_2 , equation (5.129). Also used to store $\log(G^*(\alpha_2, \beta_2) + C)$ for (5.144).

R2CON: $r_2(\psi(\beta_j)) * \beta_{j+1}$, (used in 5.150).

BDB2: β / β_{j+1} , used in (5.150).

R1: $r_2(\alpha_1, \beta_1)$ for (5.146).

R2: $r_2(\alpha_2, \beta_2)$ for (5.146).

A: $(r_2(\alpha_1, \beta_1) - r_2(\alpha_2, \beta_2)) / r_2(\alpha_1, \beta_1)$ for (5.146). Also used to store C for (5.144).

Error Messages

- (i) 'GSTAR ... ***** IS ILLEGAL': The point (α, β) is such the $RK > K_{lim}$.
- (ii) 'BETA OUT OF RANGE IN GSTAR': The value of β is greater than the maximum node for $\psi(\beta)$, (i.e. $\tilde{\tau}_f$).

5.4.4. Special considerations when including virgin risk

The term in square brackets in equation (5.63) is included only when the VIRGIN switch is true. This term, $\tilde{t}_1 \cdot r_1(\alpha)$ is non-zero for $R_{\min} < \alpha \leq R_{\max}$ and zero for $\alpha > R_{\max}$. This means that the function $G^*(\alpha, \beta)$, and hence $G(\alpha, \beta, n_0)$ can be discontinuous across the line $\alpha = R_{\max}$.

In FBET, this discontinuity is avoided by breaking the integration over α into two sections, $(\alpha_{\min}, R_{\max})$ and $[R_{\max}, \alpha_{\max}]$. As initialised, the loss factor interpolation table provides correct estimates of the loss factor for integrations over the first section. However, to comply with the requirements of integrations over the second section, the term $\tilde{t}_1 \cdot r_1(\alpha)$ must be removed from the line of table entries corresponding to $\alpha = R_{\max}$.

This facility is provided by the subroutine GADJST which adjusts the line of table entries according to the α section being integrated.

SUBROUTINE GADJST(NO)

Function

Subroutine GADJST adjusts the loss factor interpolation table to enable the integrations made by function FBET to correctly cope with the discontinuity that can exist across the line $\alpha = R_{\max}$ when the term representing failures of uncracked structures is included.

Parameter List

NO: Control parameter.

>0, the term $\tilde{t}_1 r_1(\alpha)$ is added to the table entries along the line $\alpha = R_{\max}$.

<0, the term $\tilde{t}_1 r_1(\alpha)$ is subtracted from the table entries on the line $\alpha = R_{\max}$.

Operation

The first time GADJST is called, the term $r_1(\alpha)\tilde{t}_1$ is evaluated (and stored in FACTOR) and the index value K1 corresponding to the line $\alpha = R_{\max}$ is determined. This index value is converted to an offset for the G array which contains the values of $LG_{j,k}$.

If NO>0, the following equation is applied to the entries in G corresponding to the index K1.

$$G(J+K1) = \text{ALOG1}(\text{FACTOR} + \text{EXP1}(G(J+K1))) \quad (5.152)$$

If NO<0, the equation

$$G(J+K1) = \text{ALOG1}(\text{EXP1}(G(J+K1)) - \text{FACTOR}) \quad (5.153)$$

is applied.

Note that no check is made by GADJST that NO changes sign with each successive call. It is up to the calling code that GADJST is used correctly.

5.5. The Evaluation of $P_F(t)$, $P_{det}(t)$, $r_s(t)$ and $r_v(t)$

5.5.1. Overview

For a given model, the reliability functions involving the most levels of integration are $P_F(t)$, $P_{det}(t)$ and $r_s(t)$ and it is appropriate to consider the evaluation of these three functions together. In fact, the a_0 integrand functions and the limit expressions for these functions are sufficiently similar that much of the calculations can be handled by a common set of subroutines and mutual consideration of these three functions becomes more than mere convenience.

When applicable, the evaluation of the virgin risk, $r_v(t)$ involves a similar number of levels of integration and some of the calculations can be made by the routines common to the three functions referred to above. It is therefore appropriate to include $r_v(t)$ (albeit as a slightly special case) with the functions described in this Section.

The four reliability functions considered here include at least one term which has the general form,

$$T(t) = \int_{a(n_1)}^{a(n_2)} p_{a_0}(a_0) \int_{\beta_1}^{\beta_2} p_{\beta}(\beta) \int_{\alpha_1}^{\alpha_2} p_{\alpha}(\alpha) I(\alpha, \beta, n_0, t) d\alpha d\beta da_0 \quad \dots (5.154)$$

(where $T(t)$ indicates the appropriate term in the reliability function and $I(\alpha, \beta, n_0, t)$ the relevant integrand function.)

This term can be considered to be composed of the following nested integrations,

$$T(t) = \int_{a(n_1)}^{a(n_2)} F_0(a_0) da_0 \quad (5.155)$$

$$\text{where } F_0(a_0) = p_{a_0}(a_0) \int_{\beta_1}^{\beta_2} F_\beta(\beta) d\beta ; \quad (5.156)$$

$$F_\beta(\beta) = p_\beta(\beta) \int_{\alpha_1}^{\alpha_2} F_\alpha(\alpha) d\alpha ; \quad (5.157)$$

$$F_\alpha(\alpha) = p_\alpha(\alpha) I(\alpha, \beta, n_0, t). \quad (5.158)$$

The form for this term, as described above, applies to the full model in which all the random variables are included. If any variable is neglected then the appropriate integration is replaced by a point evaluation of the integrand and the associated density function is not included. For example if a_0 is held constant, a_{con} say,

$$T(t) = F_0(a_{\text{con}}) \quad (5.159)$$

$$\text{and } F_0(a_{\text{con}}) = \int_{\beta_1}^{\beta_2} F_\beta(\beta) d\beta \quad (5.160)$$

The code structure required to perform the integrations (neglecting random variables where necessary) can be represented schematically by the flow chart in Figure 5.16 which has

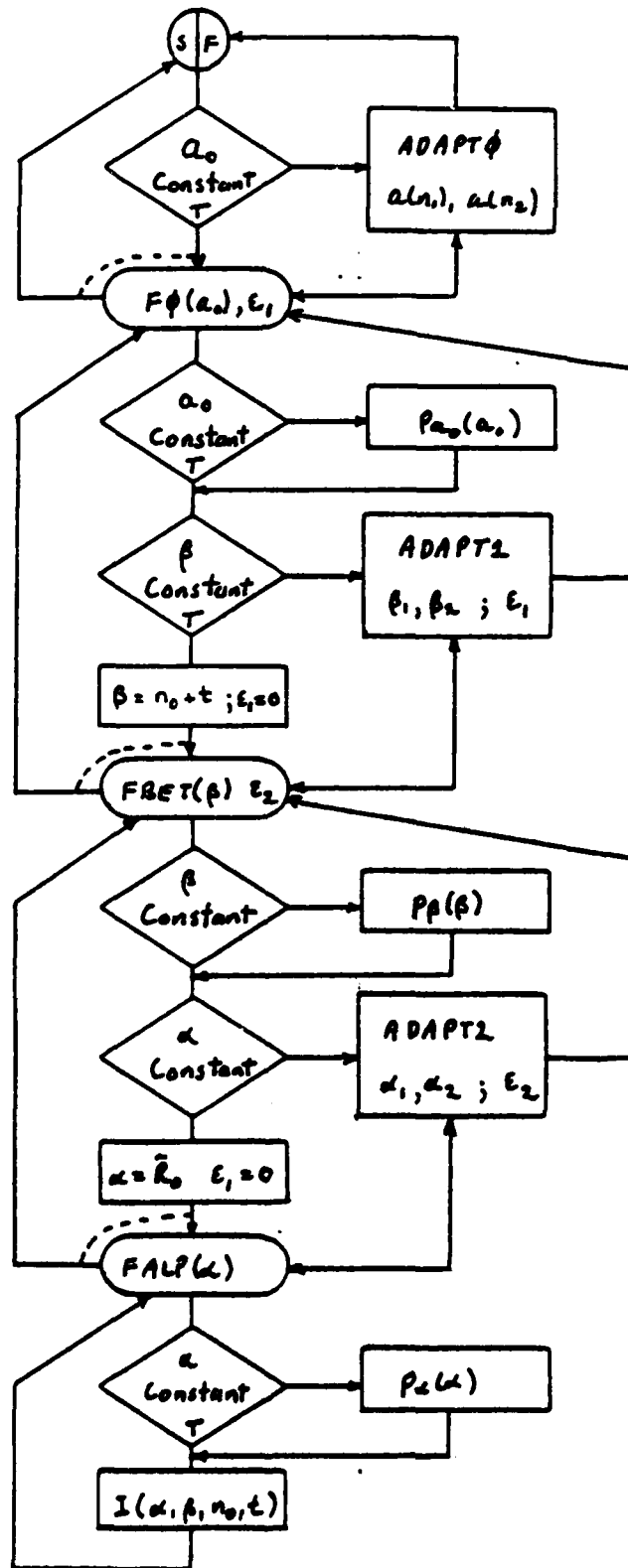


Figure 5.16. Code structure required to perform multiple integrations. See text for interpretation.

been constructed according to the following scheme.

- (i) Logic flow can be either forward (top to bottom) or reverse (bottom to top).
- (ii) Initial entry and final exit from the logic is at the top of the diagram.
- (iii) Small boxes indicate evaluation of the term within the box, only when the logic flow through the box is forward.
- (iv) On completion of the evaluation of the function in the lowermost box the logic flow direction is reversed.
- (v) Large boxes indicate the adaptive integration control routines which can direct logic flow in both directions. Integration limits are indicated and are assumed to be evaluated each time the box is entered from above.
- (vi) Diamonds indicate decisions which can affect logic flow only when approached from above. Reverse flow passes through these diamonds as if they did not exist.
- (vii) Rounded boxes indicate control points associated with the evaluation of the three levels of integrand

functions ($F_0(a_0)$, $F_\beta(\beta)$ and $F_\alpha(\alpha)$). Logic between two control points is associated with the upper control point and is included in the routine which bears the name included in the box.

The control points have memory in that reverse flow is directed along the same path as the box was entered during the previous forward flow. This concept is reinforced by using two-way arrows to indicate the links between the adaptive integration routines and the integrand function control points. This implication is that the integrand evaluation is under the control of the adaptive integration process. A one-way arrow terminating at a control point indicates a single evaluation of that integrand function. A broken line links that arrow with the return flow following evaluation of the integrand.

The diagram also indicates the flow of information relevant for the estimation of errors associated with the adaptive integration procedure. The error estimation process which was described in Section 4.1, relies on the integrand functions producing error estimates for the evaluation of the integrands. In a nested process these error estimates are produced in turn by the embedded adaptive integration routines used to evaluate the integrand functions.

The inclusion of an error variable as an argument of an integrand function control point indicates that it is the function of the logic below that point to produce that error

term which will be used by the adaptive integration algorithms above the control point. The overall error estimate for the evaluation of the reliability function is produced by the outermost adaptive integration routine.

The code used to evaluate all the reliability functions follows the general procedure described by Figure 5.16, with minor modifications (such as the omission of levels of integration or the inclusion of extra terms) as the need arises. The descriptions that follow highlight these differences and provide specifications for the integrand function routines, which, in general, will differ between reliability functions.

For the functions described in this Section, the only integrand function routine to change between functions is the outermost routine, denoted in Figure 5.16 by $F\phi$. FBET and FALP are common routines which access various logical switches and control variables to ensure that the correct forms of the integrand functions for a given reliability function are evaluated. The relevant variables and the meanings of the values that they may have are listed in Table 5.9. Note that if a single value of a logical switch is listed, then the other possible value ('true' or 'false') has the opposite meaning. Note also that the last three variables in the table are associated with graphics operations which are described in Section 5.10.

Because the switch BETCON is used to denote that \underline{X}_1 is constant and to all intents and purposes β is also constant, this case will be sometimes referred to as ' β constant'.

Name	Type	COMMON Block	Value	Meaning
ALPCON	Logical	INTCON	True	Variations in strength (σ) are ignored.
BETCON	Logical	INTCON	True	Variations in ^{relative} fatigue life are ignored, β is constant.
RNOCON	Logical	INTCON	True	Variations in initial crack length are ignored.
POPLOS	Logical	LOGCOM	False	The effect of the loss factor is ignored.
FULSV	Logical	LOGCOM	True	$P_S(t)$ is calculated via the integral expression for $P_F(t)$.
			False	$P_S(t)$ is calculated via approximate integration of $r(t)$.
RISK	Logical	LOGCOM	True	Current reliability function is $r_s(t)$.
VIRGIN	Logical	LOGCOM	True	Current model includes risk terms for uncracked structures.
INSPTS	Logical	LOGCOM	True	Current reliability functions are affected by a previous inspection.
CONTON	Logical	INTCON	True	Contour plots of integrands are required.
PLTINT	Logical	INTCON	True	Integrand plotting is activated.
NOUT	Integer	INTCON	1, 2 or 3	Level of outermost integration. e.g. 1 implies that the outermost integration is that controlled by ADAPTO in Figure 5.

Table 5.9. Variables associated with the control of the logic for the evaluation of the reliability functions.

5.5.2. Probability of Failure $P_F(t)$

The probability of failure is given by equation (3.80),

$$P_F(t) = \int_{a(n_1)}^{a(n_2)} P_F(t, n_0) p_{a_0}(a_0) da_0 + \bar{P}_{a_0}(a(n_2)) \quad (5.161)$$

where $P_F(t, n_0)$ is defined by equations (3.93), (3.116), (3.135) and (3.159) depending on the model class. These equations are summarised in Table 5.10.

Using the notation of equations (5.154) to (5.158) and using $FRLTO(a_0)$ to replace $FO(a_0)$, the set of equations determining $P_F(t)$ are equivalent to the system,

$$FRLTO(a_0) = [p_{a_0}(a_0)] \left\{ \int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} F_\beta(\beta) d\beta + \int_{\tilde{t}_f^*(R_{min})}^{\beta_{max}} p_\beta(\beta) d\beta \right\} \quad (5.162)$$

$$F_\beta(\beta) = [p_\beta(\beta)] \left\{ \int_{\alpha^*(n_0, \beta)}^{\alpha(\beta, R_{max})} F_\alpha(\alpha) d\alpha + \int_{\alpha_{min}}^{\alpha^*(n_0, \beta)} p_\alpha(\alpha) d\alpha \right\} \quad (5.163)$$

$$F_\alpha(\alpha) = [p_\alpha(\alpha)] (1 - H(\alpha, \beta, n_0, t)), \quad (5.164)$$

where, for the present, the terms representing the failures of uncracked structures have been neglected. The square brackets around the density functions indicate that the functions are included only when variations of the associated random variables are included in the model.

Probability of Failure

$$P_F(t) = \int_{a(n_1)}^{a(n_2)} P_F(t, n_0) p_{a_0}(a_0) da_0 + \bar{P}_{a_0}(a(n_2))$$

Model	$P_F(t, n_0)$
Full	$\int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} p_\beta(\beta) \left[\int_{\alpha^*(n_0, \beta)}^{\alpha_2(\beta, R_{max})} (1 - H(\alpha, \beta, n_0, t)) p_\alpha(\alpha) d\alpha + P_\alpha(\alpha^*(n_0, \beta)) \right] d\beta$ $+ \bar{P}_\beta(\tilde{t}_f^*(R_{min})) + \delta(n_0, 0) \int_{\alpha_v}^{\alpha_{max}} p_\alpha(\alpha) (1 - \exp\{-r_1(\alpha)t\}) d\alpha + P_\alpha(\alpha_v) \Big]$ $\times P_\beta(\tilde{t}_1)$
α Const.	$\int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} p_\beta(\beta) (1 - H(R_0, \beta, n_0, t)) d\beta + \bar{P}_\beta(\tilde{t}_f^*(R_{min}))$ $+ \delta(n_0, 0) (1 - \exp\{-r_1(R_0)t\}) P_\beta(\tilde{t}_1)$
β Const. $\beta = n_0 + t$	$\int_{\alpha^*(n_0, \beta)}^{\alpha_2(\beta, R_{max})} p_\alpha(\alpha) (1 - H(\alpha, t + n_0, n_0, t)) d\alpha + P_\alpha(\alpha^*(n_0, n_0 + t))$ $\tilde{t}_1 \leq t + n_0 < \tilde{t}_f$ <p>or</p> $\delta(n_0, 0) \int_{\alpha_v}^{\alpha_{max}} (1 - \exp\{-r_1(\alpha)t\}) p_\alpha(\alpha) d\alpha, t + n_0 < \tilde{t}_1$
\tilde{t}_1 and α Const.	$1 - H(\tilde{R}_0, t + n_0, n_0, t), \quad \tilde{t}_1 \leq t + n_0 < \tilde{t}_f$ <p>or</p> $1 - \exp\{-r_1(\tilde{R}_0)t\} \quad n_0 = 0 \text{ and } t < \tilde{t}_1$

Table 5.10. Summary of expressions for the probability of failure, $P_F(t)$

Equations (5.162) to (5.164) together with the logic described by Figure 5.17 are equivalent to the set of equations presented in Table 5.10 (neglecting the uncracked terms). The Figure follows the conventions established in Section 5.5.1 and includes the extra terms which represent structures that have reached the D_3 subspace, (or time-zone of failed or rejected structures).

The logic above the FRLTO control point is contained with the function RSKTOT which is described in Section 5.2.2. The remainder of the logic is handled by the adaptive integration routines together with the functions FRLTO, FBET, FALP, PBETA and PALPHA. A summary hierarchy of these functions is presented in Figure 5.18 .

The contributions from uncracked structures are included as correction terms by the function FRVO, described in Section 5.5.4 below.

When evaluating $P_E(t)$, the logical switches, RISK and FULSV have the values 'false' and 'true' respectively. These switches are used by the functions FBET and FALP to ensure that the correct integrand terms are evaluated.

The accumulation of the error estimates is largely controlled by the adaptive integration routines as described in Section 4.1.5.

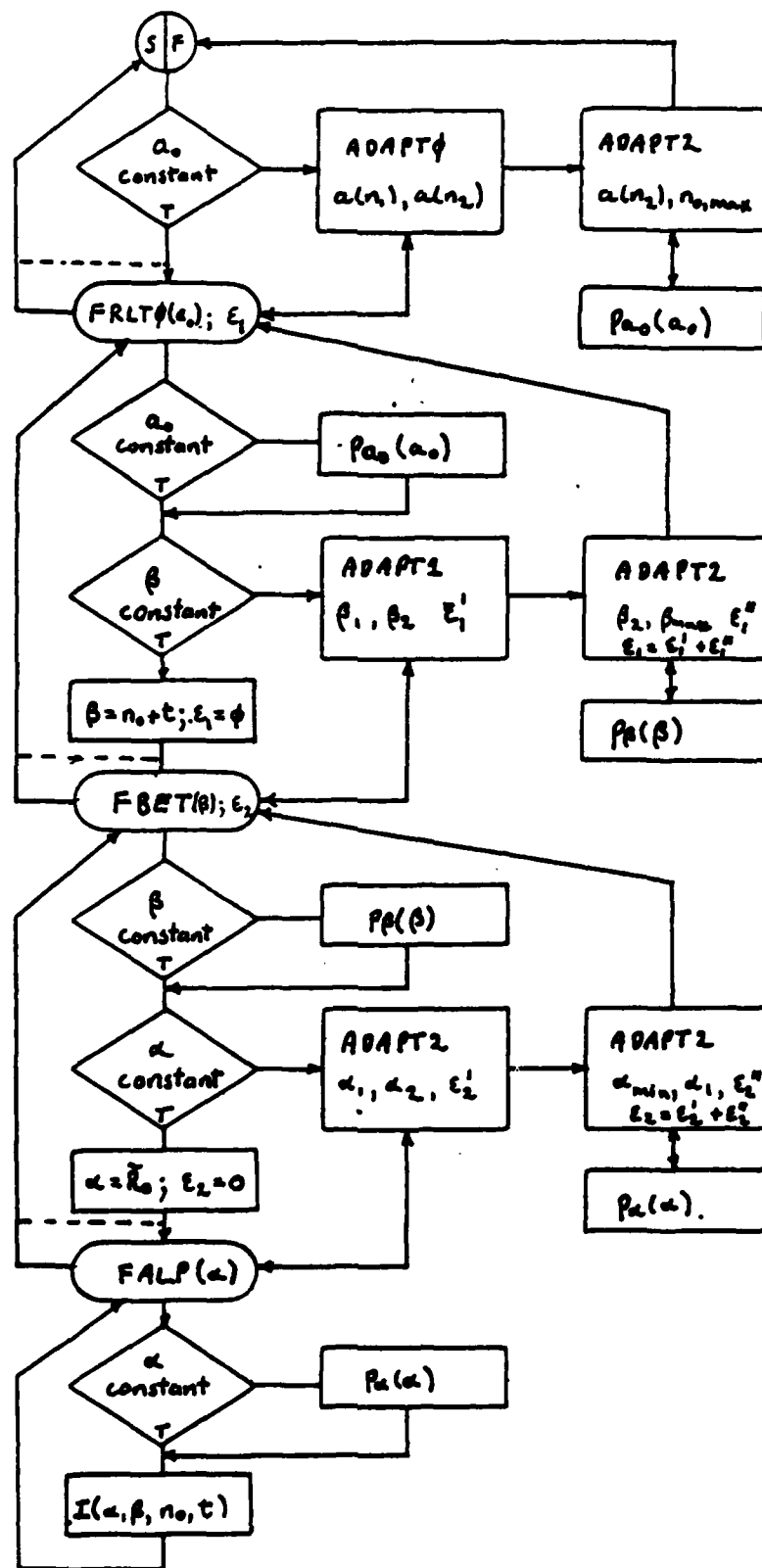


Figure 5.17. Logic associated with the evaluation of $P_F(t)$

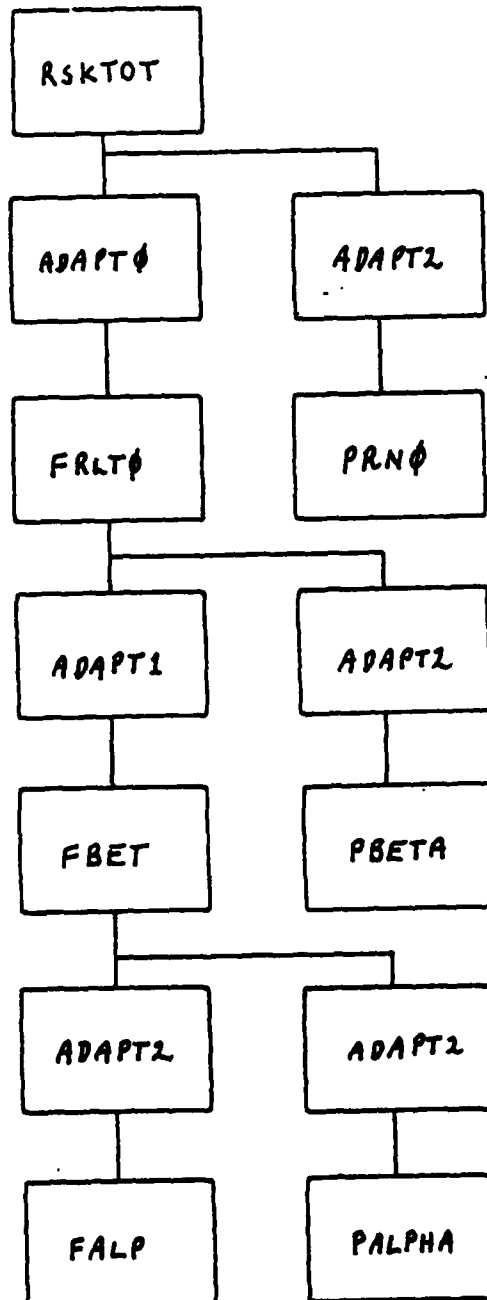


Figure 5.18. Hierarchy of function sub-programs used to evaluate $P_F(t)$.

5.5.3. Risk of Static Fracture by Fatigue

The risk of static fracture by fatigue is given by equation (3.84), i.e.,

$$r_s(t) = \int_{a(n_f)}^{a(n_2)} p_{fs}(t, n_0) p_{a_0}(a_0) da_0 / P_s(t), \quad (5.165)$$

together with equations (3.90), (3.119), (3.138) and (3.162) for the various classes. These equations are summarised in Table 5.11 and can be replaced by the set,

$$\begin{aligned} \text{FPSO}(a_0) &\equiv p_{fs}(n_0, t) p_{a_0}(a_0) \\ &= \left[p_{a_0}(a_0) \right]_{\beta_1(n_0)}^{\bar{t}_f^*(R_{\min})} F_\beta(\beta) d\beta, \end{aligned} \quad (5.166)$$

$$F_\beta(\beta) = \left[p_\beta(\beta) \right]_{\alpha(n_0, \beta)}^{\alpha_2(\beta, R_{\max})} F_\alpha(\alpha) d\alpha \quad (5.167)$$

$$\text{and } F_\alpha(\alpha) = r_2(\alpha, \gamma(\beta)) H(\alpha, \beta, n_0, t) [p_\alpha(\alpha)]. \quad (5.168)$$

together with the logic illustrated in Figure 5.16. (with FPSO replacing FO). Noting that (5.166) is equivalent to the firsts term in (5.162) $r_s(t)$ can be evaluated using FRLTO FBET and FALP using the switch RISK to remove the extra terms as required. The function hierarchy is shown in Figure 5.9.

The logic above the control point FRLTO is controlled by the function RSKTOT.

Risk of Static Failure by Fatigue

$$r_s(t) = \int_{a(n_1)}^{a(n_2)} p_{fs}(t, n_0) p_{a_0}(a_0) da_0 / P_S(t)$$

Class	$p_{fs}(t, n_0)$
Full	$\int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} p_\beta(\beta) \int_{\alpha(n_0, \beta)}^{\alpha(\beta, R_{max})} r_2(\alpha, \beta) H(\alpha, \beta, n_0, t) p_\alpha(\alpha) d\alpha d\beta$
α Const.	$\int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} p_\beta(\beta) r_2(\tilde{R}_0, \beta) H(\tilde{R}_0, \beta, n_0, t) d\beta$
χ_1 Const.	$\int_{\alpha(n_0, n_0+t)}^{\alpha(t+n_0, R_{max})} r_2(\alpha, n_0+t) H(\alpha, n_0+t, n_0, t) p_\alpha(\alpha) d\alpha$ $\tilde{t}_1 \leq n_0+t \leq \tilde{t}_f$
α and χ_1 Const.	$r_2(\tilde{R}_0, t+n_0) H(\tilde{R}_0, t+n_0, n_0, t) \quad \tilde{t}_1 \leq n_0+t < \tilde{t}_f$

Table 5.11. Summary of expressions for the risk of static fracture, $r_s(t)$.

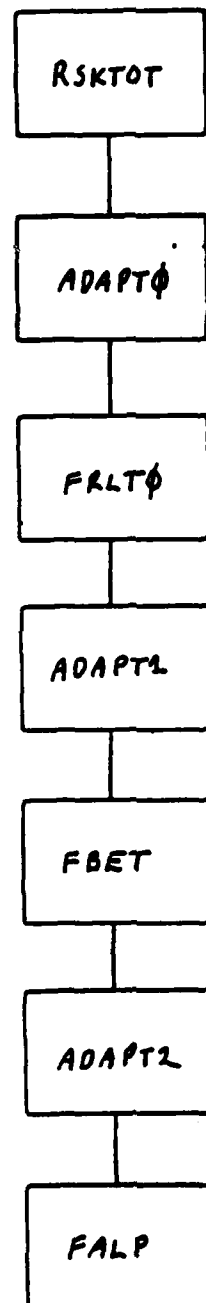


Figure 5.19. Hierarchy of function sub-programs used to evaluate $r_s(t)$.

5.5.4. Probability of Detection $P_{det}(t)$

The integral expression for the probability of detection is used only when $P_S(t)$ is computed by integrating $r(t)$ with respect to time. This condition corresponds to the switches RISK and FULSV having the values 'false' and 'false' respectively. $P_{det}(t)$ is given by equation (3.82),

$$P_{det}(t) = \int_{a(n_{d1})}^{a(n_2)} P_{det}(t, n_0) p_{a_0}(a_0) da_0 \quad (5.169)$$

together with equations (3.94), (3.117), (3.136) and (3.160) for the various classes of model. Neglecting, for the moment, the terms in (3.94) and (3.136) for uncracked structures, these equations (which are summarised in Table 5.12), can be replaced by the set,

$$\begin{aligned} FDET0(a_0) &\equiv P_{det}(t, n_0) p_{a_0}(a_0) \\ &= [p_{a_0}(a_0)] \left[\int_{\beta_1(n_0)}^{\tilde{t}_d} F_{\beta}(\beta) d\beta + \int_{\tilde{t}_d}^{\tilde{t}_f^*(R_{min})} F'_{\beta}(\beta) d\beta \right] \quad (5.170) \end{aligned}$$

$$F_{\beta}(\beta) = [p_{\beta}(\beta)] \int_{\alpha(n_0, \beta)}^{\alpha(\beta, R_p)} F_{\alpha}(\alpha) d\alpha \quad (5.171)$$

$$F'_{\beta}(\beta) = [p_{\beta}(\beta)] \int_{\alpha(n_0, \beta)}^{\alpha_{max}} F_{\alpha}(\alpha) d\alpha \quad (5.172)$$

$$F_{\alpha}(\alpha) = [p_{\alpha}(\alpha)] H(\alpha, \beta, n_0, t) \quad (5.173)$$

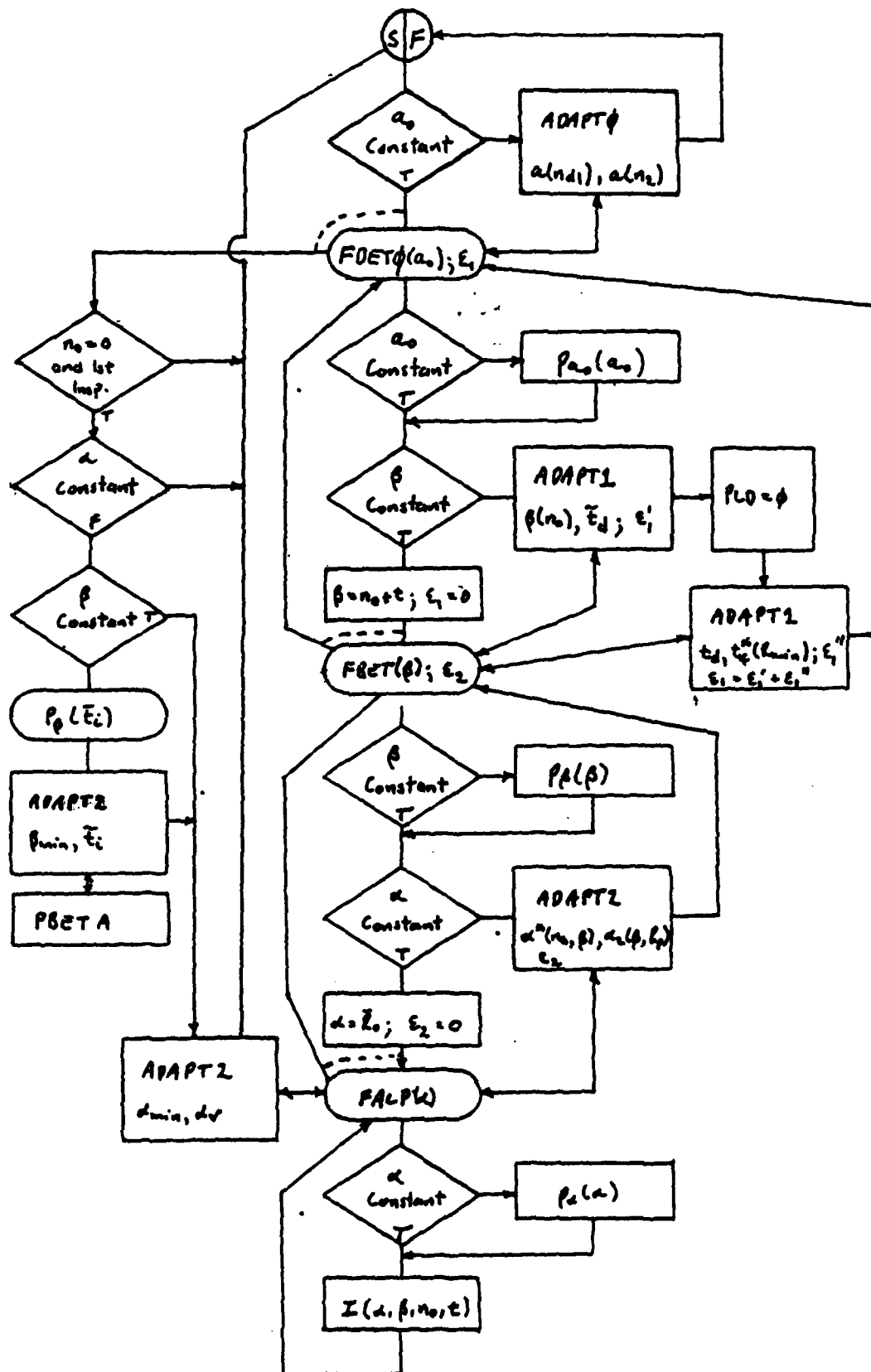
and the logic described by Figure 5.20.

Probability of Detection

$$P_{\text{det}}(t) = \int_{a(n_{d1})}^{a(n_2)} P_{\text{det}}(t, n_0) p_{a_0}(a_0) da_0$$

Class	$P_{\text{det}}(t, n_0)$
Full	$\int_{\beta_1(n_0)}^{\tilde{t}_d} p_{\beta}(\beta) \int_{\alpha^*(n_0, \beta)}^{\alpha_2(\beta, R_p)} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta$ $+ \int_{\tilde{t}_d}^{\tilde{t}_f^*(R_{\min})} p_{\beta}(\beta) \int_{\alpha^*(n_0, \beta)}^{\alpha_{\max}} p_{\alpha}(\alpha) H(\alpha, \beta, n_0, t) d\alpha d\beta$ $\delta(t, t_1) \delta(n_0, 0) P_{\beta}(\tilde{t}_1) \int_{\alpha_{\min}}^{\alpha_{\max}} \exp\{-r_1(\alpha) t\} p_{\alpha}(\alpha) d\alpha$
α Const.	$\int_{\tilde{t}_d}^{\tilde{t}_f^*(R_{\min})} p_{\beta}(\beta) H(\tilde{R}_0, \beta, n_0, t) d\beta$
χ_1 Const.	$\int_{\alpha^*(n_0, n_0+t)}^{\alpha_2(t+n_0, R_p)} p_{\alpha}(\alpha) H(\alpha, t+n_0, n_0, t) d\alpha \quad \tilde{t}_d \leq t+n_0 < \tilde{t}_f$ $\text{or } \delta(n_0, 0) \delta(t, t_1) \int_{\alpha_{\min}}^{\alpha_{\max}} p_{\alpha}(\alpha) \exp\{-r_1(\alpha) t\} d\alpha \quad t+n_0 < \tilde{t}_1$
α and χ_1 Const.	$H(\tilde{R}_0, t+n_0, n_0, t) \quad \tilde{t}_d \leq n_0+t < \tilde{t}_f$

Table 5.12. Summary of expressions for $P_{\text{det}}(t)$.

Figure 5.20. Logic associated with the evaluation of $P_{det}(t)$.

The function hierarchy is shown in Figure 5.21.

The functions $F_\beta(\beta)$ and $F'_\beta(\beta)$ differ only in the upper limit of the α integration. The function FBET detects that the probability of detection is being evaluated and sets the upper limit according to the value of $PLD, (R_p)$. If $PLD=0$, the function $F'_\beta(\beta)$ is evaluated, whereas if $PLD>0$, the function $F_\beta(\beta)$ is evaluated. Thus when the second integral in (5.170) is evaluated, PLD is set temporarily to zero and restored to its proper value after the integration has been completed. Note that in the case of a non-proof load inspection, PLD will be zero anyway: only one of the terms in (5.170) should be evaluated. The code ensures that this is the case (for simplicity, this logic is not shown in Figure 5.20)).

For models in which α is allowed variation, a term for uncracked structures must be evaluated and has the form,

$$T_v(t, n_0) = [P_\beta(\tilde{t}_1)] \int_{\alpha_{\min}}^{\alpha_v} F_\alpha(\alpha) d\alpha . \quad (5.174)$$

This extra term is included only at the first inspection.

The logic above the FDET0 control point in Figure 5.20 is controlled by the function FPDET.

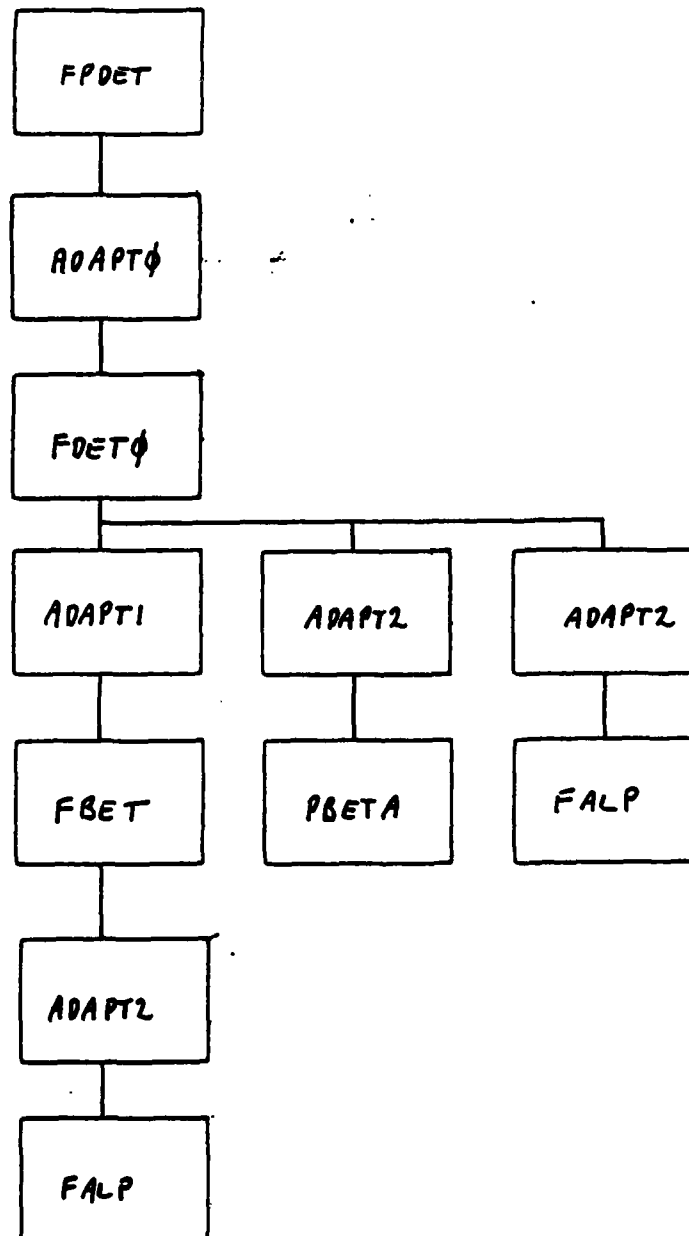


Figure 5.21. Hierarchy of function sub-programs used to evaluate $P_{det}(t)$.

5.5.5. Virgin Risk and Adjustment of $P_S(t)$

(1) Risk calculation

The virgin risk is computed only when $n_0 = 0$ and $\bar{t}_1 > 0$. These conditions can be established during data initialisation and the requirement for the calculation of $r_v(t)$ indicated by setting the switch VIRGIN 'true'. It is possible to over-ride these conditions and set VIRGIN equal to 'false' as part of the input data and thereby neglect the risk of failure of uncracked structures.

The virgin risk is given by equation (3.95), i.e.,

$$r_v(t) = \int_{\alpha_v}^{\alpha_{2(0, R_{\min})}} p_{\alpha}(\alpha) r_1(\alpha) \exp\{-r_1(\alpha)t\} d\alpha P_{\beta}(\bar{t}_1) / P_S(t). \quad \dots\dots (5.175)$$

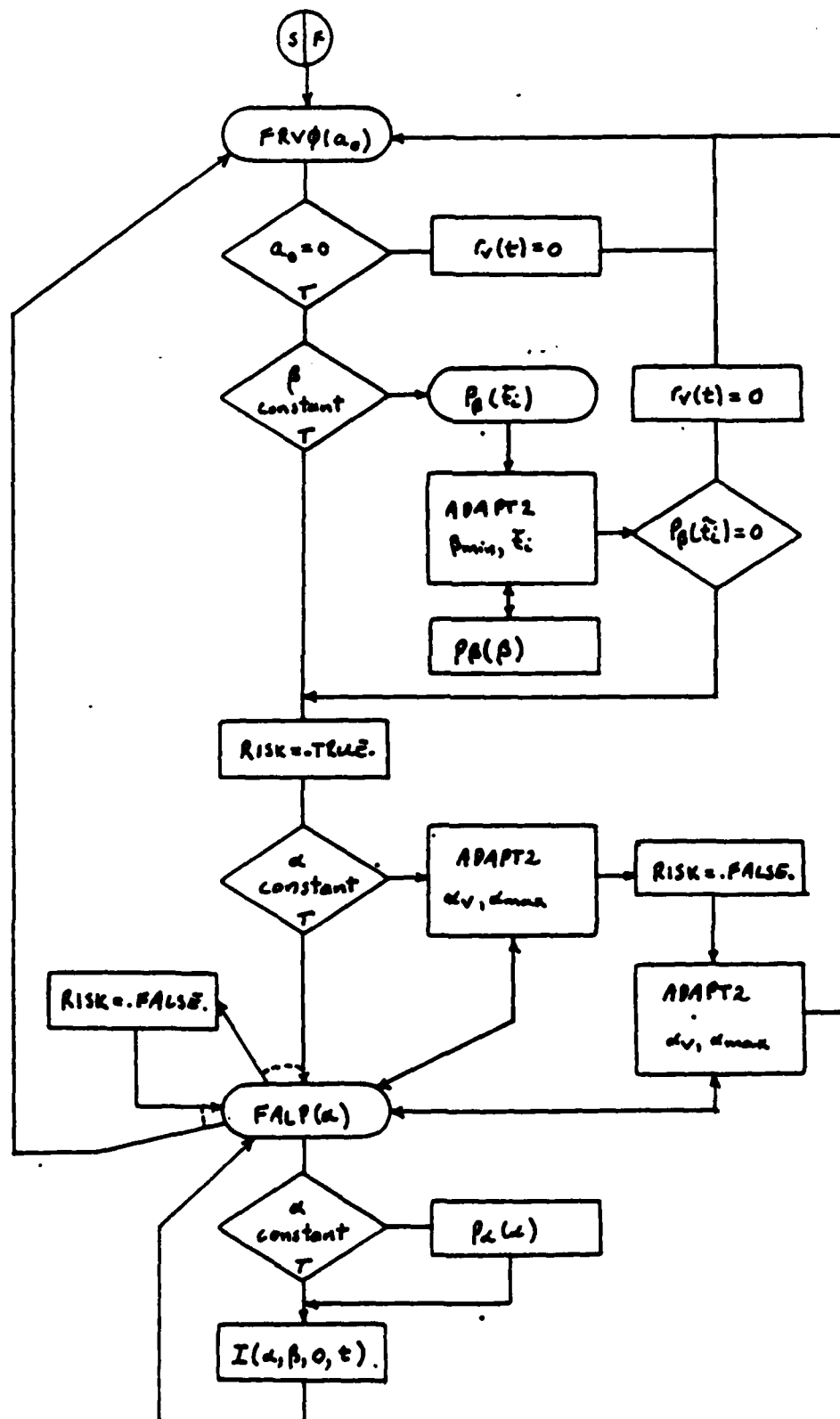
Defining the following functions,

$$\begin{aligned} FRVO(a_0) &\equiv P_{FV}(t) \cdot P_S(t) \\ &= [P_{\beta}(\bar{t}_1)] \int_{\alpha_v}^{\alpha_{2(0, R_{\min})}} F_{\alpha}(\alpha) d\alpha \end{aligned} \quad (5.176)$$

$$F_{\alpha}(\alpha) = [p_{\alpha}(\alpha)] r_1(\alpha) \exp\{-r_1(\alpha)t\}, \quad (5.177)$$

the virgin risk can be evaluated using the logic shown in Fig. 522 with RISK = 'true'. The expressions for $r_v(t)$ evaluated by this logic are summarised in Table 5.13.

Note that the integrand function $F_{\alpha}(\alpha)$ can be written

Figure 5.22. Logic associated with the evaluation of $r_v(t)$.

Virgin Risk

$$r_v(t) = \delta(n_0, 0) p_{fv}(t) / P_S(t)$$

Model	$p_{fv}(t)$
Full	$\int_{\alpha_v}^{\alpha_2(0, R_{\max})} p_{\alpha}(\alpha) r_1(\alpha) \exp\{-r_1(\alpha)t\} d\alpha P_{\beta}(\tilde{t}_1)$
α Const.	$r_1(\tilde{R}_0) \exp\{-r_1(\tilde{R}_0)t\} P_{\beta}(\tilde{t}_1)$
\tilde{X}_1 Const	$\int_{\alpha_v}^{\alpha_2(0, t)} p_{\alpha}(\alpha) \exp\{-r_1(\alpha)t\} r_1(\alpha) d\alpha \quad t < \tilde{t}_1$
α and \tilde{X}_1 Const.	$r_1(\tilde{R}_0) \exp\{-r_1(\tilde{R}_0)t\} \quad t < \tilde{t}_1$

Table 5.13. Summary of expressions for the virgin risk, $r_v(t)$

as,

$$F_{\alpha}(\alpha) = [p_{\alpha}(\alpha)] r_1(\alpha) H(\alpha, 0, 0, t) \quad (5.178)$$

so that provided the loss factor function, GVAL, returns the term $r_1(\alpha)$ for the case $n_0 = 0$, and $\beta = 0$, the function FALP can be used for both $r_s(t)$ and $r_v(t)$.

(11) Correction of $P_s(t)$

The factor $P_{\phi}(\tilde{t}_1)$ in (5.176), which involves a single level of integration, is fixed for a given value of time. Since this factor also appears in the terms in $P_s(t)$ representing uncracked structures it is computationally efficient to account for these terms when this factor is available, rather than evaluate it again when evaluating $P_s(t)$. The correction of $P_s(t)$ is therefore made by the function FRWD. Examining equation (3.93) and denoting the uncorrected probability of survival by $P_s^i(t)$, the correct function is given by, (for the full model),

$$P_s(t) = P_s^i(t) - \int_{\alpha_v}^{2(0, R_{\max})} p_{\alpha}(\alpha) (1 - \exp\{-r_1(\alpha)t\}) d\alpha - P_{\alpha}(\alpha_v) [P_s(\tilde{t}_1)] \quad \dots (5.179)$$

Denoting the correction term by $T_v(t)$ say,

$$T_v(t) = [P_\beta(\tilde{t}_1)] \left\{ \int_{\alpha_v}^{\alpha(0, R_{\max})} F_\alpha(\alpha) d\alpha \cdot P_\alpha(\alpha_v) \right\} \quad (5.180)$$

where

$$F_\alpha(\alpha) = R_\alpha(\alpha) [1 - H(\alpha; 0, 0, t)]. \quad (5.181)$$

The correction term can be evaluated using the logic in Figure 5.22, with RISK equal to 'false'.

The function hierarchy for the evaluation of $r_v(t)$ and the survivorship correction term is shown in Figure 5.23.

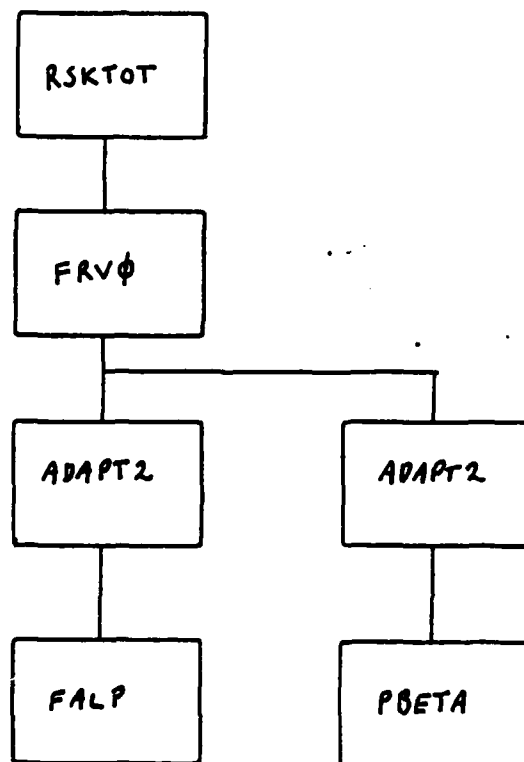


Figure 5.23. Hierarchy of function sub-programs used for the evaluation of $r_v(t)$.

FUNCTION FALP(ARG)

Function

FALP evaluates the term,

$$T(\alpha) = p_{\alpha}(\alpha) r_2(\alpha \psi(\beta)) L_f(\alpha, \beta, n_0, t) \quad (5.182)$$

where $p_{\alpha}(\alpha)$ is the density function for α , $r_2(\alpha \psi(\beta))$ is the risk rate for cracked structures and $L_f(\alpha, \beta, n_0, t)$ is a loss factor term, the precise form of which depends on the reliability function currently being evaluated.

The term evaluated by FALP is the innermost integrand of the expressions for $P_F(t)$, $P_{det}(t)$ and $r_S(t)$.

Parameter List

ARG: The value of α for which the term is to be evaluated.

Operation

The expression evaluated depends on the states of the logical switches, RISK, FULSV, POPLOS and ALPCON according to the following scheme.

If ALPCON is true, then variations in α are ignored by the model and $p_{\alpha}(\alpha)$ is replaced by 1. Otherwise, $p_{\alpha}(\alpha)$ is evaluated using the function PALPHA.

If RISK is true, then the risk term $r_2(\alpha \psi(\beta))$ is included. The current value of R is evaluated, stored in COMMON and used as the argument of function RLOAD to evaluate the risk term. If RISK is false, this risk term is not included.

If POPLOS is false, then the loss factor term is neglected from the calculation. Otherwise it is evaluated and takes the following forms.

411

- (i) If RISK is false and FULSV is true then the current calculation can only be for $P_E(t)$ and the loss factor term is given by

$$L_f(\alpha, \beta, n_0, t) = 1 - H(\alpha, \beta, n_0, t) \\ = 1 - \text{EXPI}(-\text{RNSV} * \text{GVAL}(\text{ALPV}, \text{BETV})) \dots (5.183)$$

where the required variables are assumed to be set by code external to FALP.

If the argument of EXPI is small ($\leq 10^{-4}$) the loss factor term is approximated by using the series expansion for the exponential function so that,

$$L_f(\alpha, \beta, n_0, t) = \text{RNSV} * \text{GVAL}(\text{ALPV}, \text{BETV}). \quad (5.184)$$

- (ii) If RISK is true or FULSV is true then either $P_{\text{det}}(t)$ or $r_s(t)$ is being evaluated and the loss factor term is,

$$L_f(\alpha, \beta, n_0, t) = H(\alpha, \beta, n_0, t) \\ = \text{EXPI}(-\text{RNSV} * \text{GVAL}(\text{ALPV}, \text{BETV})) \quad (5.185)$$

- (iii) If $\beta = 0$, then FALP is required to return the correct integral term for uncracked structures. It is assumed that GVAL and PSI will return $r_1(\alpha)$ and 1 respectively for this case.

Graphics Operations

If FALP is being called by the integrand contouring code, PLTPNT is called to construct a representation of the current evaluation point in (α, β) space.

If integrand functions are being plotted and the α integration is the outermost integration, PLTSTR is called to store the evaluation for later plotting as an outer integration plot.

COMMON Variables Accessed

PSIVAL: Current value of $\psi(\beta)$.

BETV: Current value of β .

ALPCON: Constant α switch.

RISK: Risk function switch.

POPLOS: Include population losses switch.

PLTINT: Plot integrands switch.

CONTON: Contour map switch.

NOUT: Outermost integration level indicator.

COMMON Variables Changed

ALPV: Current value of α .

RVAL: Current value of R .

FUNCTION FBET(ARG,ERROR1)

Function

FBET evaluates the term,

$$T(\beta) = [p_\beta(\beta)] \left[\int_{\alpha_1}^{\alpha_2} F_\alpha(\alpha) d\alpha + \int_{\alpha_3}^{\alpha_4} p_\alpha(\alpha) d\alpha \right] \quad (5.186)$$

where, numerically, $F_\alpha(\alpha) = \text{FALP}(\text{ALPHA})$. This term appears in the expressions for $P_F(t)$, $P_{\text{det}}(t)$, $r_v(t)$ and $r_g(t)$. FBET accesses various control switches to determine the appropriate expressions to evaluate.

Parameter List

ARG: Value of β for which the term is to be evaluated.

ERROR1: Error estimate associated with the evaluation of the term (returned to the calling code). This estimate is produced by the adaptive algorithms used by FBET.

Operation

The sequence of operations follows the flow chart shown in Figure 5.24. and consists of two major phases, the evaluation of $p_\beta(\beta)$ followed by the evaluation of the integral term in the square brackets in equation (5.186).

(1) Evaluation of $p_\beta(\beta)$

The value of β determined by ARG is transferred to common storage as BETV for use by other functions and is then tested to ensure that it is within the interval $[\tilde{t}_1, \tilde{t}_f]$. FBET returns the value 0 if β is outside this interval.

The value of $\psi(\beta)$ is also calculated and transferred to PSIVAL in common. If $\psi(\beta) = 0$, an error message is created and no further processing is done.

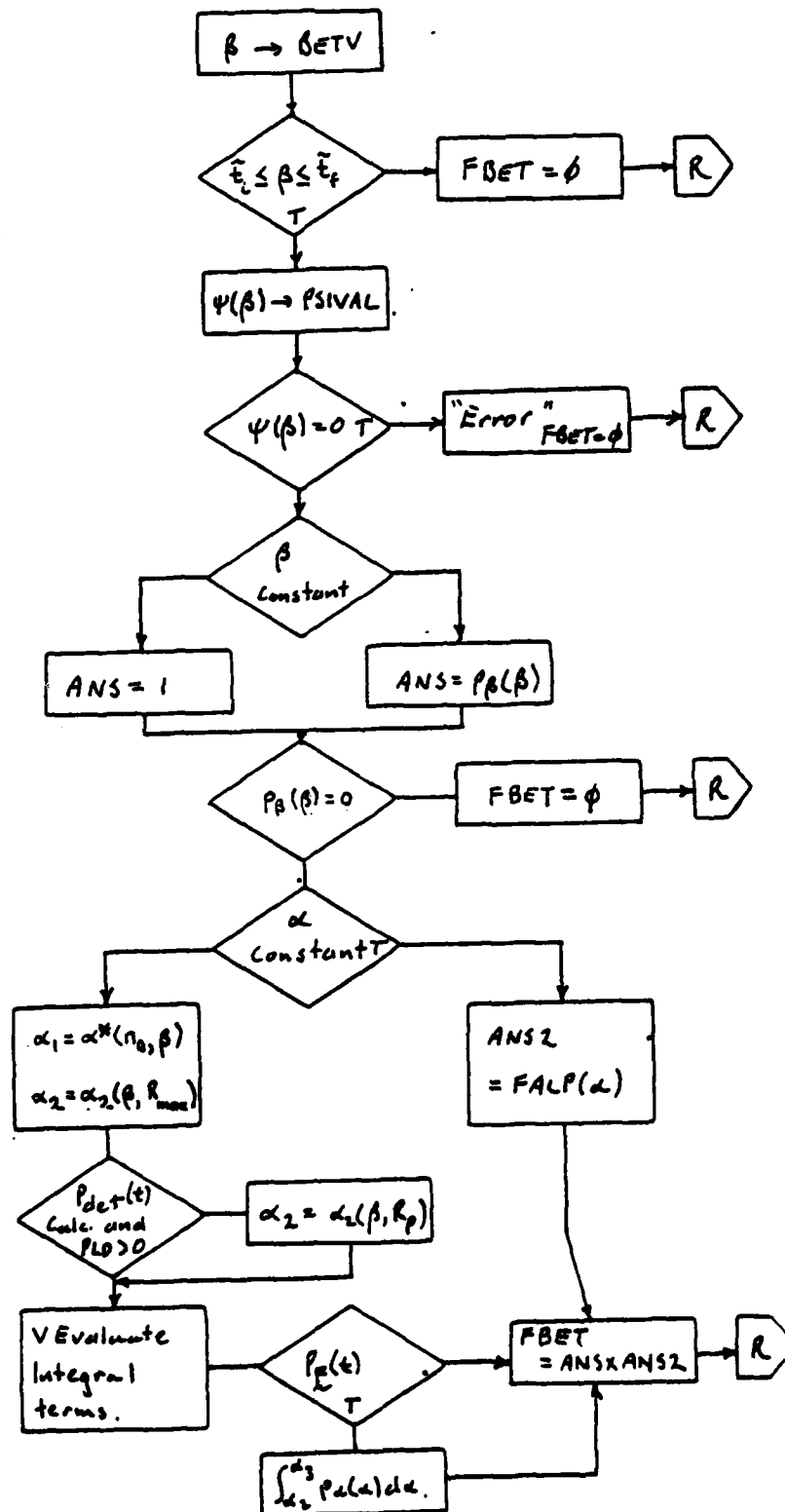


Figure 5.24. Sequence of operations executed by FBET.

If BETCON is false, the term $p_g(q)$ is evaluated using the function PBETA. If the density function is zero, no further processing is done and a zero value is returned for FBET. For a model which ignores variations in fatigue life, BETCON is true and $p_g(\beta)$ is omitted from the term returned by FBET.

(ii) Evaluation of Integrations over α

If variations in α are ignored, ALPCON will be true and a point evaluation of the integrand is made. For $\alpha < \alpha_2$, $I_\alpha(\alpha) = \text{FALP}(\text{ALPHA})$, otherwise a zero is returned.

If ALPCON is false, the integrations in (5.186) must be evaluated. The limits α_1 , α_2 and α_3 are evaluated according to the requirements of the function being evaluated by the external code. For all functions,

$$\alpha_1 = \alpha^*(n_0, \beta) \quad (5.187)$$

as defined by equations (3.52), (3.53) and (3.128). For $P_F(t)$ and $r_g(t)$,

$$\alpha_2 = \alpha_2(\beta, R_{\max}) \quad (5.188)$$

as defined by equation (3.56), while for $P_{\text{det}}(t)$

$$\alpha_2 = \alpha_2(\beta, R_p) \quad (5.189)$$

or α_{\max} if R_p is zero (see Section 5.5.4).

The second term exists only when $P_F(t)$ is being evaluated; for this function,

$$\alpha_3 = \alpha_{\min} \quad (5.190)$$

If the reliability model includes a virgin risk term, it is possible that the loss factor term, which is included in the integrand term evaluated by FALP, is discontinuous across the line $R_{\max} = \alpha$. The handling of this discontinuity is discussed in Section 5.4.4 in

connection with the evaluation of the loss factor. The essential requirement as far as EFFT is concerned is that the integration of FALP must be made over two sub-intervals $[\alpha_1, p_{\max}]$ and $[p_{\max}, \alpha_2]$. A call to GADIST before and after the integration over the second interval modifies the interpolation table appropriately.

The code in EFFT performs the necessary integrations and accumulates the error estimates from each integration in a straightforward and obvious manner using the adaptive algorithms described in Section 4.1.

Graphics Operations

If the integration over β is the outermost integration of the function being evaluated and integrand plotting is required (as controlled by PLTINT), PLTSTP is called to store each evaluation for later plotting.

Error Messages

- (1) 'EFFT PSI IS ZERO FOR BETA = *****'; EFFT has been called with APC equal to a value of β for which $\psi(\beta)$ is zero. No further processing is made and EFFT returns the value 0.

Common Variables Accessed

- PMF: Current value of $\tilde{\epsilon}_f$.
- PMI: Current value of $\tilde{\epsilon}_1$.
- P(1): Current value of p_{\min} .
- P(N): Current value of p_{\max} .
- PLD: Current value of p_j .

RNO: Current value of n_0 .

RNIV: Time of last inspection.

RNSV: Current value of time.

ALPMIN: α_{\min} .

ALPMAX: α_{\max} .

ALPCON: Constant α switch.

BETCON: Constant β switch.

INSPTS: Inspection switch.

POPLOS: Include population losses switch.

VIRGIN: Virgin risk switch.

EPSINT: Convergence criterion for adaptive integration.

NOUT: Outermost integration level indicator.

COMMON Variables Changed

BETV: Current value of β .

PSIVAL: Current value of $p(\beta)$.

FUNCTION FDETO(ARG,ERROR1)

Function

FDETO evaluates the function $P_{det}(t, n_0)$ used in the evaluation of the probability of detection. This term has the form, for the most general model,

$$FDETO(a_0) = [p_{a_0}(a_0)] \left\{ \int_{\beta_1(n_0)}^{\tilde{t}_d} F_{\beta}(\beta) d\beta + \int_{\tilde{t}_d}^{\tilde{t}_f^{(R_{min})}} F_{\beta}'(\beta) d\beta \right\}$$

$$\delta(t, t_1) \delta(n_0, 0) < P_{\beta}(\tilde{t}_1) \int_{\alpha_{min}}^{\alpha} \exp \{ -r_1(\alpha) \} p_{\alpha}(\alpha) d\alpha \}$$

..... (5.191)

Parameter List

ARG: Value of a_0 for which the evaluation of $P_{det}(t, a^{-1}(a_0)) [p_{a_0}(a_0)]$ is required.

ERROR1: Error estimate, returned by FDETO, representing the accumulated effects of any integrations used in the evaluation.

Operation

FDETO is evaluated in two stages. The first stage follows the logic between the control points FDETO and FBET in Figure 5.20 and evaluates FDET without any contribution from uncracked structures.

The second stage, which is applicable only at the first inspection, evaluates the contribution from uncracked structures which are rejected by a proof load inspection.

The code is a straightforward implementation of the requirements of the various possibilities listed in Table 5.12.

Note the use of PLD to switch FBET to use alternative limits of integration. The correct value of PLD is stored in PLDSTR while PLD is temporarily set to zero. Note that if $PLD = 0$ on entry to FDET0, only one of the integrations over β is made.

FUNCTION PPDET(PINSP)

Function

PPDET evaluates the probability of detection, $P_{det}(t)$ at an inspection.

Parameter List

PINSP: Value of time at which $P_{det}(t)$ is required.

Operation

PPDET is called only when the probability of survival is calculated via approximate integration of the total risk rate and handles the logic above the PPDET control point in Figure 5.20.

For the case of models with varying a_0 , it is assumed that the call to PPDET always follows a call to PSUTOT for the same value of time as specified by PINSP, so that the a_0 limits, n_1 and n_2 are available in RNOMIN and RNOLIM respectively. For the case of a model with constant relative fatigue life, RNOMIN is modified to yield n_{d1} as specified by equation (3.14.8), (see table 5.16). The a_0 limits are then evaluated and the adaptive routine ADAPTO used to perform the required integration.

Graphics Operations

The construction of an outer integrand plot is controlled by calling PLTSET on entry and PLTOFF on completion of the evaluation of $P_{det}(t)$.

COMMON Variables Changed

RNOMIN: Lower a_0 limit for integration.

FUNCTION FRLTO(ARG)

Function

FRLTO evaluates the functions $[p_{a_0}(a_0)]P_F(t, n_0)$, (RISK=false) or $[p_{a_0}(a_0)]P_F(t, n_0)$, (RISK=true). The function returned by FRLTO has the general form,

$$FRLTO(a_0) = [p_{a_0}(a_0)] \left\{ \int_{\beta_1(n_0)}^{\tilde{t}_f^*(R_{min})} F_\beta(\beta) d\beta + \int_{\tilde{t}_f^*(R_{min})}^{\beta_{max}} p_\beta(\tilde{t}_f^*(R_{min})) d\beta \right\}$$

.... (5.192)

Parameter List

ARG: Value of a_0 for which the evaluation is required.

Operation

FRLTO controls the logic between the control points FRLTO and FBET in Figure 5.17 or F0 and FBET in Figure 5.16. A detailed flow chart for the function is given in Figure 5.25 which also includes the graphics operations executed by FRLTO.

The following points are worthy of notice.

- (i) B1 and B2 contain the limits $\beta_1(n_0)$ and $\tilde{t}_f^*(R_{min})$ respectively and these are evaluated by the subroutine FBETALY.
- (ii) The integrations with respect to β use the nodes of loss factor interpolation table as a basis for adaptive integration strategy when appropriate.

Graphics Operations

If required, PLTSTP is called to store the evaluation for subsequent construction of an outer integrand plot.

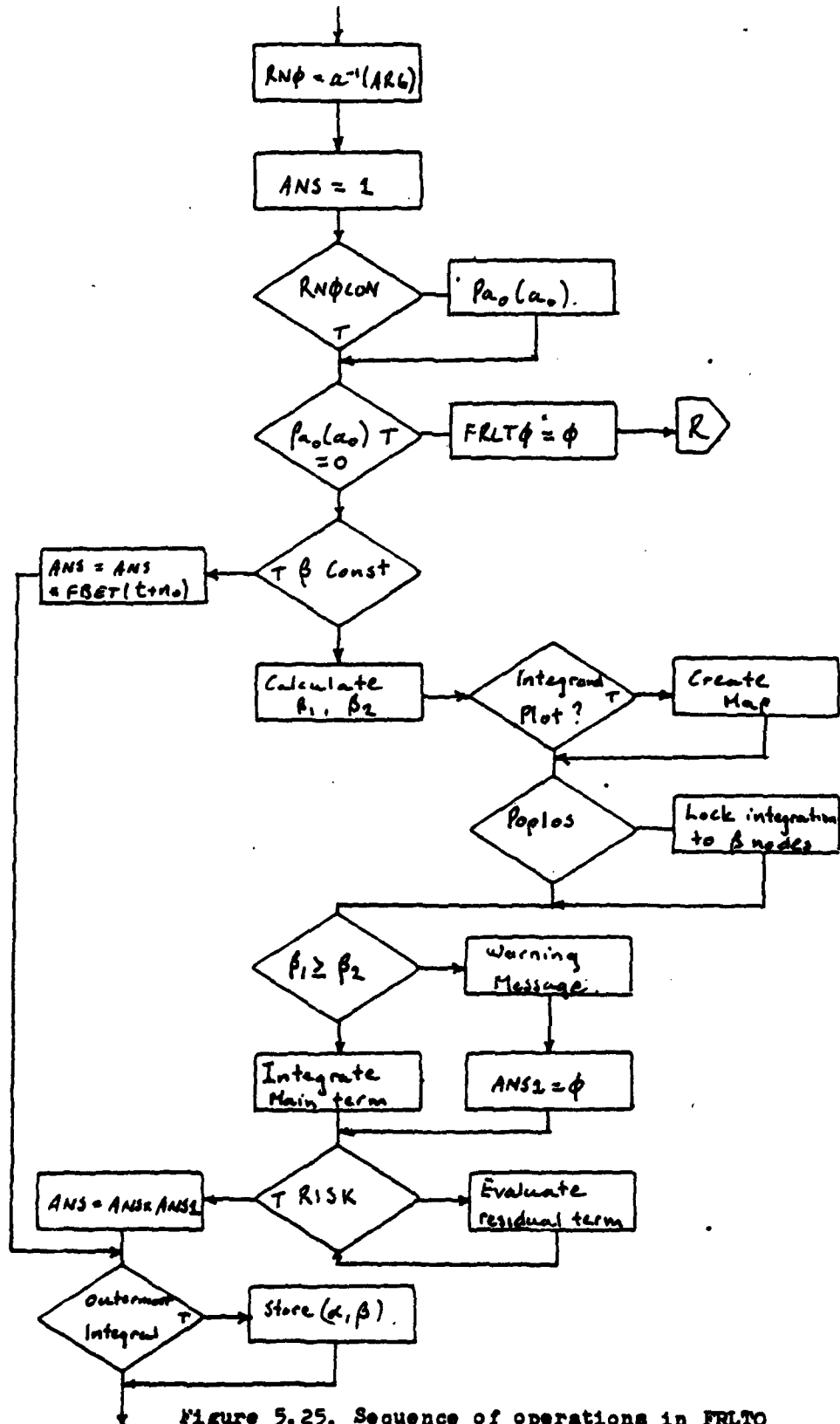


Figure 5.25. Sequence of operations in FRLTO

The construction of two-dimensional contour maps (in α, β space) of the integrand functions is controlled by FRLT0. The whole process is controlled by the subroutine INTPLT, as described in Section 5.10.2.

Error Messages

- (i) 'FRLT0 ... B1.GE.B2'; The limits of integration preclude an evaluation of the first term in curly brackets. The error is non-fatal and FRLT0 returns a zero value for this term.

COMMON Variables Changed

RN0: Current value of n_0 .

FUNCTION FRVO(ARG)

Function

FRVO controls the evaluation of the virgin risk rate, $r_v(t)$ and, if necessary, the adjustment of $P_S(t)$ to account for failures of uncracked structures.

Parameter List

ARG: Value of a_0 for which the virgin risk is required. ARG must be zero otherwise the calculation of virgin risk is not appropriate for the model.

Operation

FRVO controls the logic above the FALP control point in Figure 5. The calculation of $r_v(t)$ is executed first by evaluating $P_S(\tilde{t}_1)$, (if BETCON is false) and then either integrating FALP over ∞ or making a point evaluation of the integrand (FALP).

The $P_S(t)$ correction term is evaluated by repeating the steps involving FALP, but with RISK set to false.

Prompts

- (i) 'VIRGIN RISK TERMS'; indicates that subsequent prompts issued by the adaptive integration routines are associated with the calculation of $r_v(t)$.
- (ii) 'VIRGIN SURVIVORSHIP TERM'; indicates that subsequent prompts issued by the adaptive integration routines are associated with the correction term for $P_S(t)$.

COMMON Variables Changed

RLT: Current value of $P_S(t)$.

5.6. Limits and Their Evaluation

The discussion in Section 5 concentrated on the evaluation of the integral expressions defining the reliability functions, in terms of the evaluation of the nested sequence of integrations and integrand functions. The logic diagrams indicated the processes involved to ensure that the correct forms of the various functions are evaluated. In these diagrams, the limits of integration were given with the appropriate adaptive integration algorithm name; the code associated with the evaluation of the limits was not described.

The various subroutines, function sub-programs and code sequences used to evaluate the integration limits are described in this Section.

The description starts with the process of establishing the default limits defined by equations (3.43) to (3.49). The limits for the outermost integration variable, a_0 , are then discussed, followed by the β and α limits.

5.6.1. Default limits

The process of establishing the default limits starts with the definition of initial values by the input data. The various definitions and program variables used to store the initial values are summarised in Table 5.14.

The initial values are gradually overwritten as the first two phases of operation are executed (Table 5.1). The default limits, the program variables containing the final values and the subroutines in which the values are set are summarised in Table 5.15.

The following points should be noted.

- (i) Primes denote initial values of the variables.
- (ii) As already mentioned in Section 3.4.1., initial crack length, a_0 , is used as the integration variable for the outermost integration, whereas initial age, n_0 , is used to represent that variable within the integrand functions. There is thus a dual relationship between a_0 and n_0 . For each limit defined for n_0 , there is a corresponding limit for a_0 .

Both sets of limits are not given in the Tables in which the limits are expressed in terms of n_0 . For each limit

Symbol	Definition of initial value	Name of variable
$n'_{0,min}$	Initial crack length density (via RANGE)	RNOMIN
$n'_{0,max}$	Initial crack length density (via RANGE) or, Input value of RNO (CNO)	RNOMAX
\tilde{t}'_1	First node of $a(\tilde{t})$	RNI
\tilde{t}'_f	Last node of $a(\tilde{t})$	RNF
\tilde{t}'_d	Specified value or, last node of $C_d(a)$	RND (CND)
R'_{min}	First node of $\bar{P}_L(R)$	R(1)
R'_{max}	Last node of $\bar{P}_L(R)$	R(M)
α'_{min}	Density for relative residual strength via RANGE.	ALPMIN
α'_{max}	Density for relative residual strength via RANGE If strength variation ignored $\alpha'_{min} = \alpha'_{max} = \bar{R}_0$.	ALPMAX
$x'_{1,min}$	Density for relative fatigue life via Range	B1 (in BETCNG)
$x'_{1,max}$	Density for relative fatigue life via RANGE	B2 (in BETCNG)

Table 5.14. Definitions of and program variables string initial values of quantities used to determine the default limits.

Symbol	Limiting operation	Name of variable	Subroutine where set
$n_{0,min}$	$\max \{ \tilde{t}_1, n_{0,min}' \}$	RNOMIN	CFIN and SETTAB
$n_{0,max}$	$\min \{ \tilde{t}_f, n_{0,max}' \}$	RNOMAX	CFIN and SETTAB
α_{min}	α_{min}'	ALPMIN	CFIN
α_{max}	α_{max}'	ALPMAX	CFIN
R_{min}	$\max \{ \alpha_{min}' \psi(\tilde{t}_f), R_{min}' \}$	R(1)	SETTAB
R_{max}	$\min \{ \alpha_{max}', R_{max}' \}$	R(M)	SETTAB
\tilde{t}_1	$\max \{ \tilde{t}_1', \psi^{-1}(R_{max}/\alpha_{min}') \}$	RNI, or BETA(1)	SETTAB
\tilde{t}_f	$\min \{ \tilde{t}_f', \psi^{-1}(R_{min}/\alpha_{max}') \}$	RNF or BETA(N)	SETTAB
\tilde{t}_d	$\min \{ \tilde{t}_d', \psi^{-1}(R_p/\alpha_{max}') \}$	RND	ADVNC
$1/x_{1,min}$	$1/x_{1,min}'$	B1 (in BETCNG)	CFIN via BETSET
$1/x_{1,max}$	$1/x_{1,max}'$	B2 (in BETCNG)	CFIN via BETSET
$t/x_{1,min}$	$t/x_{1,min}'$	BETMIN	BETCNG
$t/x_{1,max}$	$t/x_{1,max}'$	BETMAX	BETCNG
β_{min}	$t/x_{1,min} + n_0$	BETMIN+RNO	} as required
β_{max}	$t/x_{1,max} + n_0$	BETMAX+RNO	

Table 5.15. Limiting operations involved in establishing the default limits. Table includes indications of the sections of code in which the operations are performed.

thus expressed, the limit for initial crack length can be obtained via the crack length - age relationship, $a(\bar{t})$.

In the code, two sets of values for the limits are stored. The variable names for initial age values all start with 'R'; those for crack length start with 'C'. For example, CNO and RNO are the names of the variables containing the current value of a_0 and n_0 respectively.

- (iii) The current version of the code assumes that the specified maximum for a_0 is less than a_d . The limit equation (3.49) is not completely executed within the code. (It was not expected that the analysis of a population of structures having initial crack lengths greater than the inspection threshold would be a real possibility.)
- (iv) The limit equations for x_1 are also included. although these are not strictly part of the set defined by equations (3.43) to (3.49)
- (v) The default limits for n_0 are applied in two steps. The first, in CFIN, uses \bar{t}_1' and \bar{t}_f' . The second, in SETTAB, uses \bar{t}_1 and \bar{t}_f . Normalisation of the density function for initial crack length is done in CFIN; the implication is that this normalisation is based on the age limits as specified by $a(\bar{t})$.

5.6.2. Limits for a_0 or n_0

The integration limits for initial crack length, a_0 , are determined for the various reliability functions by specified limiting values for initial age, n_0 . These limiting values are defined in Section 3.4.1. For a given model, the limits for all the reliability functions can be reduced to a set of 7 values; n_1 , n_2 , n_{d1} , n_{f1} , n_{f2} , n_{R1} , and n_{R2} . The equations defining these values for all the models were presented in Section 3.4 and are summarised in Table 5.16.

The limits for the functions described in Section 5.5 are determined by the three values n_1 , n_2 and n_{d1} , and are evaluated by code in RSKTOT which controls the n_0 integrations for $r_s(t)$ and $P_s(t)$ and FPDET which controls the n_0 integration for $P_{det}(t)$. The values n_{f1} and n_{f2} are used for the risk of fatigue life exhaustion, $r_f(t)$ and are evaluated by code in RSKTOT. The values n_{R1} and n_{R2} are evaluated by code in FLPROB which controls the n_0 integrations for the strength functions.

The equations summarised in Table 5.16, rely on three functions f_{n_0} , $f_{n_0}(R)$ and $\beta'_{p,R}$ which are handled in the code as described below.

Limit	Full model and constant relative strength	Constant rel. fatigue life	Constant rel. fatigue life and strength
n_1	$n_{O,min}$	$n_{O,min}$	$n_{O,min}$
n_2	$\min\{n_{O,max}, (t \cdot \tilde{t}_d - \beta_{min} t_{i_j}) / (t - t_{i_j})\}$	$\min\{n_{O,max}, \tilde{t}_f^*(R_{min}) - t\}$	$\min\{n_{O,max}, \tilde{t}_f^*(R_{min}) - t\}$
n_{f_1}	$n_{O,min}$	$\max\{n_{O,min}, \beta'_{p,R_{min}} - t\}$	$\tilde{t}_f - t$
n_{f_2}	$\min\{n_{O,max}, \max\{f_{n_O}, f_{n_O}(R_{min})\}\}$	n_2	$\tilde{t}_f - t$
n_{d1}	$n_{O,min}$	$\max\{n_1, \tilde{t}_d - t\}$	$\tilde{t}_d - t$
n_{R1}	$n_{O,min}$	$\max\{n_1, \beta'_{p,R} - t\}$	$\psi^{-1}(R/\tilde{R}_O) - t$
n_{R2}	$\min\{n_{O,max}, f_{n_O}(R)\}$	n_2	$\psi^{-1}(R/\tilde{R}_O) - t$

Table 5.16. Summary of equations for the integration limits for initial age, or initial crack length.

(i) f_{n_0}

The function f_{n_0} , which is used to determine n_{f2} for the full model only, is given by,

$$f_{n_0} = (\bar{t}_d t - \bar{t}_f t_{1j}) / (t - t_{1j}), \quad (5.193)$$

(see equation 3.78).

In terms of program variables, (5.193) is equivalent to,

$$f_{n_0} = (RND \cdot RNSV - RNF \cdot RNIM) / (RNSV - RNIM). \quad (5.194)$$

(ii) $f_{n_0}(R)$

The function $f_{n_0}(R)$ is defined by equation (3.72) for the most general model and (3.114) for a model with constant relative strength. For the latter case, $f_{n_0}(R_{\min})$ is the same as f_{n_0} .

This function is evaluated by the function sub-program called FNOB described below.

(iii) $\beta'_{p,R}$

The function $\beta'_{p,R}$ is given by the solution of the equation,

$$R = R_p \psi(\beta) / \psi(\beta_1(n_0, \beta)). \quad (5.195)$$

$\beta'_{p,R}$ is returned as one of the arguments of the subroutine ALPHAP which also computes $\alpha'_{p,R}$ given by

$$\alpha'_{p,R} = R/\gamma(\beta'_{p,R}) . \quad (5.196)$$

ALPHAP uses the function FRP which evaluates the right hand side of equation (5.195).

SUBROUTINE ALPHAP(ALPP,BETAP,PV,R1,R2)

Function

ALPHAP finds the lower limit for α for an integration along the line $R=\alpha\psi(\beta)$, subject to restrictions on the permissible range of β .

Parameter List

ALPP: α_p ; α coordinate of the lower limit of integration.

BETAP: β_p ; β value corresponding to α_p .

RV: Value of R for which α_p and β_p are required.

R1: Lower bound for β_p .

R2: Upper bound for β_p .

Operation

The values of α_p and β_p returned by ALPHAP depend on the juxtaposition of the lines $\beta=\beta_1$, $\beta=\beta_2$, $\alpha=R/\psi(\beta)$ and $\alpha=R_p/\psi(\beta_1(\beta,n_0))$ where the function $\beta_1(\beta,n_0)$ depends on the class of model. For models accounting for variations in fatigue life,

$$\beta_1(\beta,n_0) = n_0 + (\beta - n_0)t_{1j}/t \quad (5.197)$$

and for those ignoring variations in fatigue life,

$$\beta_1(\beta,n_0) = \beta - t + t_{1j}. \quad (5.198)$$

This function is assumed to be available via function FRP() which returns,

$$FRP(\beta) = \psi(\beta)R_p/\psi(\beta_1(\beta,n_0)) \quad (5.199)$$

for given β .

The values of α_p and β_p are determined according to the following conditions.

- (i) If $R_p < R$, corresponding to $PLD < RV$, the lower limit of integration is determined by $\beta = \beta_1$ and the returned values are,

$$\alpha_p = R_p / \psi(\beta_1) ; \beta_p = \beta_1. \quad (5.200)$$

- (ii) If $R_p / \psi(\beta_1(\beta_1, n_0)) \leq R / \psi(\beta_1)$, corresponding to $FRP(\beta_1) \leq R$, the lower limit of integration is again determined by $\beta = \beta_1$ and the returned values are the same as for condition (i).

- (iii) If $R_p / \psi(\beta_1(\beta_2, n_0)) \geq R / \psi(\beta_2)$, corresponding to $FRP(\beta_2) \geq R$, the lower limit is on the line $\beta = \beta_2$. In this case β_p is returned as β_2 but α_p is returned as $R_p / \psi(\beta_1(\beta_2, n_0))$ which represents the intersection of $\alpha = R_p / \psi(\beta_1(\beta_2, n_0))$ and $\beta = \beta_2$.

- (iv) Otherwise, the limit is found by solving the equation

$$R = \psi(\beta) R_p / \psi(\beta_1(\beta, n_0)) \quad (5.201)$$

using FSOLVE and appropriate initial guesses.

FUNCTION FNOR(RV)

Function

FNOR evaluates the function $f_{n_0}(R)$ used to determine the upper limit for n_0 for integral expressions for which the restriction $R = \alpha\psi(\beta)$ holds.

Parameter list

RV: Value of R for which $f_{n_0}(R)$ is required.

Operation

The function $f_{n_0}(R)$ is defined by equation (3.72),

$$f_{n_0}(R) = \min \left\{ \beta_{\max}, \psi^{-1}(R/\alpha_{\max}), \left[t \cdot \min \{ \tilde{t}_d, \psi^{-1}(R_p \psi(\beta')/R) \} \right. \right. \\ \left. \left. - \min \{ \beta', \max \{ \psi^{-1}(R/\alpha_{\min}), \psi^{-1}(R \psi(\tilde{t}_d)/R_p), \beta_{\min} \} \} t_{1j} \right] \right. \\ \left. / (t - t_{1j}) \right\} \\ \dots (5.202)$$

where

$$\beta' = \min \{ \tilde{t}_f, \beta_{\max}, \psi^{-1}(R/\alpha_{\max}) \} \quad (5.203)$$

or, if α is constant equation (3.114),

$$f_{n_0}(R) = \min \{ \psi^{-1}(R/\tilde{R}_0), (\tilde{t}_d t - \psi^{-1}(R/\tilde{R}_0) t_{1j}) / (t - t_{1j}) \} . \\ \dots (5.204)$$

If there have been no inspections, or $t = t_{1j}$, equations (5.202) and (5.204) are replaced by,

$$f_{n_0}(R) = \beta' . \quad (5.205)$$

FUNCTION FRP(ARG)

Function

FRP evaluates the function

$$f(\beta) = R\psi(\beta) / \psi(\beta_1(\beta, n_0)) \quad (5.206)$$

which is used to determine the lower limits of integration along the line $R = \alpha\psi(\beta)$.

Parameter List

ARG: Value of β for which the function is required.

Operation

The function $\beta_1(\beta, n_0)$ depends on the status of the BETCON switch. If BETCON = .FALSE.,

$$f(\beta) = R\psi(\beta) / \psi(n_0 + (\beta - n_0)t_{1j}/t) \quad (5.207)$$

and if BETCON = .TRUE.,

$$f(\beta) = R\psi(\beta) / \psi(\beta - t + t_{1j}). \quad (5.208)$$

If, in the latter case, $\beta > \bar{\epsilon}_f + t - t_{1j}$, $f(\beta) = 0$.

5.6.3. Limits for β

The limits for β are given in the integral expressions listed in the tables of reliability functions described in this Chapter (e.g. Table 5.10 for $P_F(t)$). The limits depend on various functions which were defined in Section 3.3.2, (see Table 3.3). The methods by which these functions are evaluated are described below.

(1) $\tilde{t}_f^*(R)$

The upper limit for an integration over β is expressed in terms of $\tilde{t}_f^*(R)$, where

$$\tilde{t}_f^*(R) = \min\{\tilde{t}_f, \beta_d(n_0), \beta_{\max}, \psi^{-1}(R/\alpha_{\max})\} . \quad \dots \quad (5.209)$$

For $R = R_{\min}$,

$$\tilde{t}_f^*(R_{\min}) = \min\{\tilde{t}_f, \beta_d(n_0), \beta_{\max}\} . \quad (5.210)$$

The function $\beta_d(n_0)$ is defined by,

$$\beta_d(n_0) = n_0 + (\tilde{t}_d - n_0)t/t_{1j} \quad (5.211)$$

$$\text{or} \quad = \tilde{t}_d - t_{1j} - t \text{ if } \lambda_1 \text{ is constant.} \quad (5.212)$$

The limit, $\tilde{t}_f^*(R_{\min})$ is returned by the subroutine BETALM. $\tilde{t}_f^*(R)$ can then be recovered using,

$$\tilde{t}_f^*(R) = \min\{\tilde{t}_f^*(R_{\min}), \psi^{-1}(R/\alpha_{\max})\} . \quad (5.213)$$

(ii) $\beta_1(n_0)$

The function $\beta_1(n_0)$ is given by

$$\beta_1(n_0) = \max \{ n_0, \beta_{\min}, \bar{\tau}_1 \}. \quad (5.214)$$

and is returned by the subroutine BETALM.

(iii) $\beta_{p,R}$

The function $\beta_{p,R}$ is defined by,

$$\beta_{p,R} = \max \{ \psi^{-1}(R/\lambda_{\min}), \beta'_{p,R}, \beta_1(n_0) \}. \quad (5.215)$$

Subroutine ALPHAP is used to obtain $\beta'_{p,R}$ and equation (5.215) is then applied to yield $\beta_{p,R}$. (see description of FLDR0).

SUBROUTINE BETALM(B1,B2)

Function

BETALM evaluates the functions $\beta_1(n_0)$ and $\tilde{t}_f^*(R_{\min})$ which are used to determine the integration limits for β .

Parameter List

B1: Lower β limit, $\beta_1(n_0)$.

B2: Upper β limit, $\tilde{t}_f^*(R_{\min})$.

Operation(i) Models which include variations in fatigue life.

For given initial crack length, a_0 , or initial age, n_0 , $\beta_1(n_0)$ is defined by equation (5.214), so that

$$B1 = \max \{ n_0, \beta_{\min}, \tilde{t}_1 \}. \quad (5.216)$$

For a given value of time, BETMIN contains the minimum value of t/x_1 so that β_{\min} is given by,

$$\begin{aligned} \beta_{\min} &= n_0 + (t/x_1)_{\min} \\ &= RNO + BETMIN \end{aligned} \quad (5.217)$$

For positive $(t/x_1)_{\min}$, B1 is thus given by,

$$\begin{aligned} B1 &= \max \{ RNO + BETMIN, RNI \} \\ &= \text{AMAX1}(RNO + BETMIN, RNI). \end{aligned} \quad (5.218)$$

The upper limit, $\tilde{t}_f^*(R_{\min})$, is given by equation (5.210), so that, neglecting previous inspections,

$$B2 = \text{AMIN1}(BETMAX + RNO, RNF) \quad (5.219)$$

If inspections are included,

$$B2 = \min \{B2, \beta_d(n_0)\} \quad (5.220)$$

where B2 on the RHS has been evaluated using (5.219).

Equation (5.220) leads to,

$$B2 = \text{AMIN1}(B2, \text{RNO} + (\text{RND} - \text{RNO}) * \text{RNSV} / \text{RNIM}).$$

..... (5.221)

(ii) Models with x_1 constant

If x_1 is constant, there are no integrations over β . However, the function $\tilde{t}_f^*(R_{\min})$ is still used to set some of the α limits. BETALM has been written to return sensible limit values for such models.

B1 is set to \tilde{t}_1 as given by RNI.

Without inspections,

$$\tilde{t}_f^*(R_{\min}) = \tilde{t}_f, \quad (5.222)$$

so that B2 is returned with the value, RNF.

If there have been inspections,

$$\begin{aligned} \tilde{t}_f^*(R_{\min}) &= \min \{ \tilde{t}_f, \beta_d(n_0) \} \\ &= \min \{ \tilde{t}_f, \tilde{t}_d - t_{1j} + t_j \}, \end{aligned} \quad (5.223)$$

so that

$$B2 = \text{AMIN1}(\text{RNF}, \text{RND} - \text{RNIM} + \text{RNSV}) \quad (5.224)$$

5.6.4. Limits for α

The integration limits for α depend on the functions $\alpha_2(\beta, R)$, $\alpha^*(n_0, \beta)$ and α_v defined in Section 3.3.2 and listed, for reference, in Table 3.3. Typical code sequences for their evaluation are described below.

(i) $\alpha_2(\beta, R)$.

The equation defining $\alpha_2(\beta, R)$ is,

$$\alpha_2(\beta, R) = \min\{\alpha_{\max}, R/\psi(\beta)\} \quad (5.225)$$

which leads to

$$\alpha_2(\beta, R_{\min}) = \text{AMIN1}(\text{ALPMAX}, R(1)/\text{PSI}(\text{BETV})) \quad (5.226)$$

in FBET, or

$$\alpha_2(\tilde{\tau}_f^*(R_{\min}), R_{\min}) = \text{AMIN1}(\text{ALPMAX}, R(1)/\text{PSI}(\text{RFBET})) \quad (5.227)$$

.....

where RFBET has been set to $\tilde{\tau}_f^*(R_{\min})$, in FRFO.

In FRVO, $\alpha_2(0, R)$ is given by,

$$\alpha_2(0, R) = \text{AMIN1}(\text{ALPMAX}, R(M)) \quad (5.228)$$

(ii) $\alpha^*(n_0, \beta)$

The equation defining $\alpha^*(n_0, \beta)$ is

$$\alpha^*(n_0, \beta) = \max \{ \alpha_{\min}, R_{\min} / \psi(\beta), R_p / \psi(\beta_1(n_0, \beta)) \} \quad (5.229)$$

....

Remembering that the function FRP(β) returns the function

$$\text{FRP}(\beta) = R_p \psi(\beta) / \psi(\beta_1(n_0, \beta)), \quad (5.230)$$

the third term in the curly brackets in (5.229) is given by $\text{FRP}(\beta) / \psi(\beta)$ and the FORTRAN equivalent of (5.229) is,

$$\alpha^*(n_0, \beta) = \text{AMAX1}(\text{ALPLOW}, R(1) / \text{PSIVAL}, \text{FRP}(\text{BETV}) / \text{PSI}(\text{BETV})) \quad (5.231)$$

...

(iii) $\underline{\alpha}_v$

The lower limit for the virgin risk α integration is given by

$$\alpha_v = \max \{ \alpha_{\min}, R_{\min}, R_p \}, \quad (5.232)$$

which is evaluated by,

$$\alpha_v = \text{AMAX1}(\text{ALPMIN}, R(1), \text{PLD}). \quad (5.233)$$

5.7. Risk of Fatigue Life Exhaustion

The risk of fatigue life exhaustion is given by equation (3.85), i.e.,

$$r_f(t) = \int_{a(n_{f1})}^{a(n_{f2})} (p_{ff1}(t, n_0) + p_{ff2}(t, n_0)) p_{a_0}(a_0) da_0 / P_S(t) \quad \dots (5.234)$$

together with equations (3.97), (3.120), (3.139) and (3.163) for $p_{ff1}(t, n_0)$ and (3.98), (3.121), (3.140) and (3.164) for $p_{ff2}(t, n_0)$ for the various classes. These equations are summarised in Tables 5.17 and 5.18.

The evaluation of the term $r_f(t) \cdot P_S(t)$ follows the logic shown in Figure 5.26. The operations above the FRFO control point are executed by code in RSKTOT.

Note that integrations over α occur only for the case of the full model. If one of α or β are held constant, $r_f(t)$ is reduced to a single integration over a_0 . If both α and β are held constant, $r_f(t)$ is given by a point evaluation of the integrand at a value of a_0 which depends on the constant values of α and β and the current value of t .

The function hierarchy is shown in figure 5.27.

Risk of Fatigue Life Exhaustion

$$r_f(t) = \int_{a(n_{f1})}^{a(n_{f2})} [p_{ff1}(t, n_0) + p_{ff2}(t, n_0)] p_{a_0}(a_0) da_0 / p_g(t)$$

Class	$p_{ff1}(t, n_0)$
Full	$\int_{R_{min}/\psi(\beta, R_{min})}^{\omega_2(\tilde{t}_f^*(R_{min}), R_{min})} p_{\omega}(\alpha) H(\alpha, \beta^*, n_0, t) (\beta^* - n_0) p_{\beta}(\beta^*) d\alpha / t$ $\left(\beta^* = \psi^{-1}(R_{min}/\alpha) \right)$
ω Const.	$p_{\beta}(\tilde{t}_f) H(\tilde{R}_0, \tilde{t}_f, n_0, t) (\tilde{t}_f - n_0) \delta(\tilde{t}_f^*, \tilde{t}_f^*(R_{min})) / t$
X_1 Const $\beta = n_0 + t$	$p_{\omega}(R_{min}/\psi(\beta)) H(R_{min}/\psi(\beta), \beta, n_0, t) R_{min} \psi'(\beta) / \psi(\beta)^2$ $\tilde{t}_1 \leq n_0 + t < \tilde{t}_f$
X_1 and ω Const.	$\delta(\tilde{t}_f, \tilde{t}_f^*(R_{min})) \delta(n_0, \tilde{t}_f - t) H(\tilde{R}_0, \tilde{t}_f, \tilde{t}_f - t, t)$

Table 5.17. Summary of expressions for $p_{ff1}(t, n_0)$.

Class	$p_{ff2}(t, n_0)$
Full	$\delta(\tilde{t}_f, \tilde{t}_f^*(R_{min})) p_g(\tilde{t}_f) \left(\frac{\tilde{t}_f - n_0}{t} \right) \int_{\alpha^*(n_0, \tilde{t}_f)}^{\alpha_{max}} p_\alpha(\alpha) H(\alpha, \tilde{t}_f, n_0, t) d\alpha$
α Const.	0
X_1 Const.	$\delta(\tilde{t}_f, \tilde{t}_f^*(R_{min})) \delta(\tilde{t}_f - t, n_0) \int_{\alpha^*(\tilde{t}_f - t, \tilde{t}_f)}^{\alpha_{max}} p_\alpha(\alpha) H(\alpha, \tilde{t}_f, \tilde{t}_f - t, t) d\alpha$
α and X_1 Const.	0

Table 5.18. Summary of expressions for $p_{ff2}(t, n_0)$.



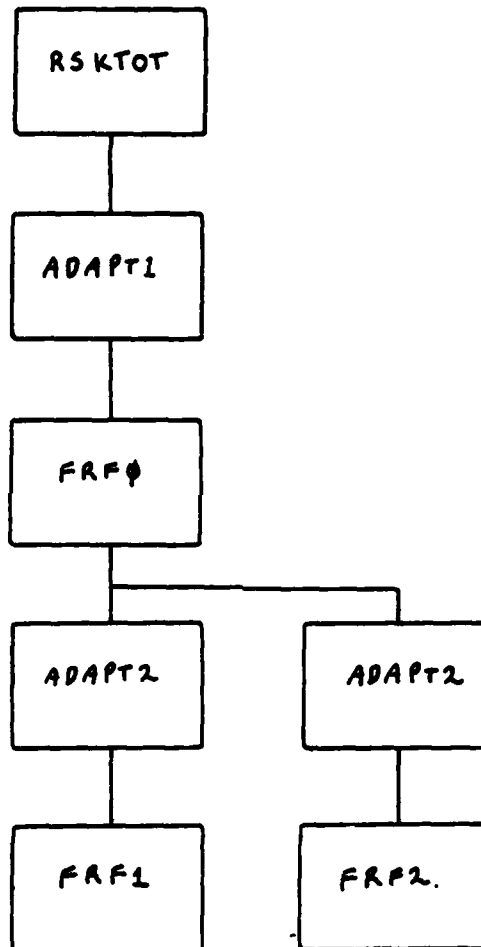


Figure 5.27. Hierarchy of function sub-programs used for the evaluation of $r_f(t)$.

FUNCTION FRF0(ARG,ERROR1)

Function

FRF0 evaluates the term

$$T(a_0) = [p_{a_0}(a_0)] (p_{ff1}(t,n_0) + p_{ff2}(t,n_0)) \quad (5.235)$$

which is used as the a_0 integrand for $r_f(t)$.

Parameter List

ARG: Value of a_0 for which the function value is required.

ERROR1: Error estimate returned by FRF0 and representing the error attributed to any integrations.

Operation

FRF0 controls the logic between the control points FRF0 and FRF1 or FRF2 in Figure 5.26 and consists of two sections of code. The first evaluates $p_{ff1}(t)$; the second evaluates $p_{ff2}(t,n_0)$. In each section, the form of the integrand depends on the states of the switches ALPCON and BETCON as outlined below.

(1) First section $p_{ff1}(t)$

Full Model

The calculation of $p_{ff1}(t)$ for models with variations in α and X_1 included involves an integration over α , which is made in the usual way using ADAPT2 and FRF1 as an integrand function.

The upper α limit is calculated by direct application of equation (5.227).

The lower limit, $R_{\min}/\psi(\beta_{p,R_{\min}})$, is calculated by the subroutine ALPHAP. Note that the call to ALPHAP restricts the possible solution for $\beta_{p,R_{\min}}$ to the interval $[\beta_1, \tilde{t}_f^*(R_{\min})]$.

α Constant, β Varying

If α is constant, $p_{ffl}(t, n_0)$ is given by

$$p_{ffl}(t, n_0) = p_\beta(\tilde{t}_f)(\tilde{t}_f - n_0)H(R_0, \tilde{t}_f, n_0, t)/t \quad (5.236)$$

and exists only when $\tilde{t}_f^*(R_{\min}) \geq \tilde{t}_f$.

Provided $RFBET \geq RNF$, the term is given by

$$p_{ffl}(t, n_0) = PBETA(RNF) * (RNF - RNO) / RNSV * (EXP(-RNSV * GVAL(ALPV, RNF))). \quad (5.237)$$

α Varying, β Constant

If β is constant and lies in the interval $[\tilde{t}_1, \tilde{t}_f]$, $p_{ffl}(t, n_0)$ is the only term that exists and is given by,

$$\begin{aligned} p_{ffl}(t, n_0) &= p_\alpha(\alpha)H(\alpha, \beta, n_0, t)R_{\min}\psi'(\beta)/\psi(\beta)^2 \\ &= PALPHA(ALPV) * R(1) * PSIDEV(BETV) / (PSIARG)^2 \\ &\quad * EXP(-RNSV * GVAL(ALPV, BETV)) \end{aligned} \quad (5.238)$$

where

$$\alpha = ALPV = R_{\min}/\psi(\beta) = R(1)/PSI(BETV) \quad (5.239)$$

and

$$\beta = BETV = n_0 + t = RNO + RNSV. \quad (5.240)$$

α Constant, β Constant

For a model with both α and β constant, the $r_f(t)$ term exists only when $\tilde{t}_f^*(R_{\min}) = \tilde{t}_f$, ($RFBET \geq RNF$) and $n_0 = \tilde{t}_f - t$, ($BETV = RNF$). The term is then given by,

$$p_{ff1}(t, n_0) = H(R_0, \bar{t}_f, \bar{t}_f - t)$$

$$= \text{EXPI}(-\text{RNSV} \# \text{GVAL}(\text{ALPV}, \text{BETV})) \quad (5.241)$$

where ALPV and BETV have been set to the values \bar{R}_0 and \bar{t}_f respectively prior to the evaluation of $r_f(t)$.

If the model allows for variations in n_0 , the answer is multiplied by da_0/dn_0 , given by CRKDEV(RNO).

(ii) Second Section, $p_{ff2}(t, n_0)$

$p_{ff2}(t, n_0)$ exists only when $\bar{t}_f^*(R_{\min}) \geq \bar{t}_f$, ($\text{RFBET} \geq \text{RNF}$) and consists of an integration, using ADAPT2, of FRF2 over α .

The lower limit, $\alpha^*(n_0, \bar{t}_f)$ is evaluated using the definition equation (3.52),

$$\alpha^*(n_0, \bar{t}_f) = \max\left\{\alpha_{\min}, R_{\min}/\psi(\bar{t}_f), R_p/\psi(\beta_1(n_0, t_f))\right\}$$

$$= \text{AMAX1}(\text{ALPMIN}, R(1)/\text{PSI}(\text{RNF}), \text{ALPP}) \quad (5.242)$$

where ALPP ($=\alpha_p$) has already been returned a previous call to ALPHAP. The logic in ALPHAP ensures that ALPP will contain the correct value of α_p which represents the intersection between the R_{\min} boundary and the line $\beta = \bar{t}_f$.

Following the integration of FRF2, the factor $n_f(\bar{t}_f)(\bar{t}_f - n_0)/t$ is included, if β is not constant, to complete the evaluation.

FUNCTION F271(ARG)

Function

F271 evaluates the integrand for the first $v_2(t)$ term. This term, which is evaluated only when α and β are both free to vary has the form (see equation (3.27),

$$t(\alpha) = (\beta^* - n_0) v_2(\alpha) H(\alpha, \beta^*, n_0, t) v_\beta(\beta^*) \quad (5.243)$$

where $\beta^* = \psi^{-1}(v_{min}/\alpha).$ (5.244)

Parameter List

ARG: Value of α for which the integrand term is to be evaluated.

Operation

The local variable F271 is used to store β^* which is computed from the value of α passed into F271 via ARG. If $\beta^* \leq n_0$, a value of zero is returned for the integrand. Otherwise the term is evaluated using the functions PALPHA and PBETA for $v_2(\alpha)$ and $v_\beta(\beta)$ respectively with GVAL being used to evaluate the loss factor, $H(\alpha, \beta^*, n_0, t)$ if required, (as indicated by the switch, POPLOS).

Graphics Operations

PLTST2 is called to store the integrand evaluation if an outer integrand plot is to be constructed.

FUNCTION FFF2(ARG)

Function

FFF2 evaluates the integrand function for the second $r_e(t)$ term. This function has the form,

$$T(\alpha) = p_\alpha(\alpha) W(\alpha, \tilde{t}_e, n_\alpha, t). \quad (5.245)$$

Parameter List

ARG: Value of α for for which the integrand function is required.

Operation

The code in FFF2 evaluates the term in a straightforward manner with the loss factor, $W(\alpha, \tilde{t}_e, n_\alpha, t)$ being omitted when losses from the population are neglected.

Graphics Operations

PLTST2 is called to store the evaluation for subsequent construction of an outer integral plot if required.

5.8. Strength Distributions

In Section 5.2.3., the organisation of the code in FLPROB which controls the evaluation of the strength related functions was described. These functions are; $p_R(R|t)$, $P_R(R|t)$ and $E(R|t)$, the density, distribution and expected value for the failing load, and $p_R(R|\underline{F} > t)$, $P_R(R|\underline{F} > t)$ and $E(R|\underline{F} > t)$, the density, distribution and expected value for strength.

The documentation for FLPROB presented in that Section described how, for a given value of time, the density functions are evaluated for a series of values of R and are then integrated over R by trapezoidal integration to yield the distributions and expected values.

This Section is concerned with the code which evaluates the two density functions. Most of this code is located within the DO LOOP which terminates at statement 100 and sweeps through the set of R values.

5.8.1. Probability density for strength

The density function for strength is given by equation (3.87), i.e.,

$$p_R(R | \underline{F} > t) = \int_{a(n_{R1})}^{a(n_{R2})} f_R(R, t, n_0) p_{a_0}(a_0) da_0 / P_S(t) \quad (5.246)$$

together with equations (3.99), (3.122), (3.141) and (3.165) which define $f_R(R, t, n_0)$ for the various classes. These equations are summarised in Table 5.19.

The evaluation the term $p_R(R | \underline{F} > t) \cdot P_S(t)$ follows the logic shown in Figure 5.28. The operations above the FLPROB control point are executed by the code in FLPROB.

The result of this calculation is stored in the local variable, TERM. It is converted to $p_R(R | \underline{F} > t)$ by dividing by $P_S(t)$, which is stored in RLTD(J), the value of which was computed during the construction of the time sequence of reliability functions and was stored by the subroutine ADVNCE.

Note that integrations over β occur only in the case of the full model. If either α or \underline{X}_1 is constant, then $f_R(R, t, n_0)$ becomes a point evaluation for a value of β which is determined by n_0 and which ever of α or \underline{X}_1 is constant.

If α and \underline{X}_1 are held constant, then the only meaningful

Probability Density for Strength

$$P_R(R|F > t) = \int_{a(n_{R1})}^{a(n_{R2})} f_R(R, t, n_0) p_{a_0}(a_0) da_0 / P_S(t)$$

Class	$f_R(R, t, n_0)$
Full	$\int_{\beta_p, R}^{\tilde{t}_f^*(R)} \frac{p_\beta(\beta) p_\alpha(\alpha) H(\alpha, \beta, n_0, t)}{\psi(\beta)} d\beta$ $(\alpha = R/\psi(\beta))$ $+ \delta(n_0, 0) H_s(R - R_p) p_\alpha(R) \exp\{-r_1(R)t\} p_\beta(\tilde{t}_1)$
α Const. $\beta = \psi^{-1}(R/\tilde{R}_0)$	$p_\beta(\beta) H(\tilde{R}_0, \beta, n_0, t) / [\tilde{R}_0 \psi'(\beta)]$ $+ \delta(n_0, 0) \delta(R, \tilde{R}_0) \exp\{-r_1(\tilde{R}_0)t\} p_\beta(\tilde{t}_1)$
\tilde{X}_1 Const $\beta = n_0 + t$ $\alpha = R/\psi(\beta)$	$p_\alpha(\alpha) H(\alpha, \beta, n_0, t) / \psi(\beta) \quad \tilde{t}_1 \leq t + n_0 < \tilde{t}_f$ $\text{or } \delta(n_0, 0) H_s(R - R_p) p_\alpha(R) \exp\{-r_1(R)t\}$ $t + n_0 < \tilde{t}_1$
α and \tilde{X}_1 Const. $\beta = n_0 + t$ $\beta = \psi^{-1}(R/\tilde{R}_0)$	$H(\tilde{R}_0, \beta, n_0, t) / [\tilde{R}_0 \psi'(\beta)]$ $\tilde{t}_1 \leq t + n_0 < \tilde{t}_f$

Table 5.19. Summary of expressions for $f_R(R, t, n_0)$.

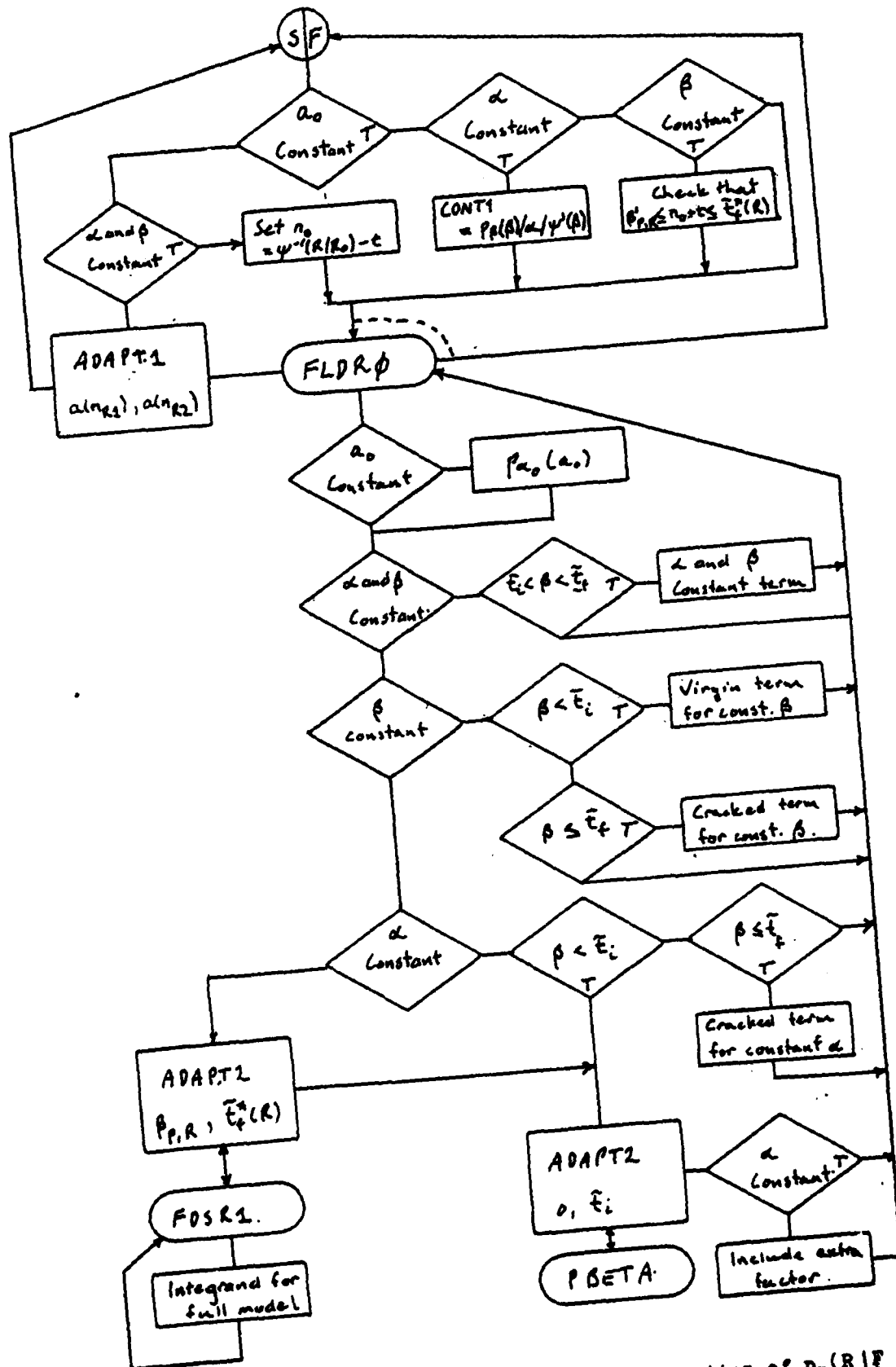


Figure 5.28. Logic associated with the evaluation of $P_R(R|F > t)$.

calculation is one for which n_0 is varying. FLPROB tests for the possibility that all random variables are held constant for which case no strength calculations are made. The correct form of $p_R(R | \underline{F} > t)$ for the case of n_0 varying and the other two random variables held constant is,

$$\begin{aligned} p_R(R | \underline{F} > t) &= f_R(R, t, n_0) p_{n_0}(n_0) / P_S(t) \\ &= f_R(R, t, n_0) p_{a_0}(a_0) \frac{da_0}{dn_0} / P_S(t) . \quad (5.247) \end{aligned}$$

The function FLDRD multiplies the function given in Table 5.19 by the factor da_0/dn_0 to meet equation (5.247).

The code in FLPROB associated with the evaluation of $p_R(R | \underline{F} > t)$ is combined with code associated with the evaluation of $p_R(R | t)$ since both functions are evaluated concurrently. Examination of the code, in conjunction with Figure 5.28 should identify the appropriate sections of code.

The function sub-program hierarchy associated with the evaluation of the density for strength is shown in Figure 5.29.

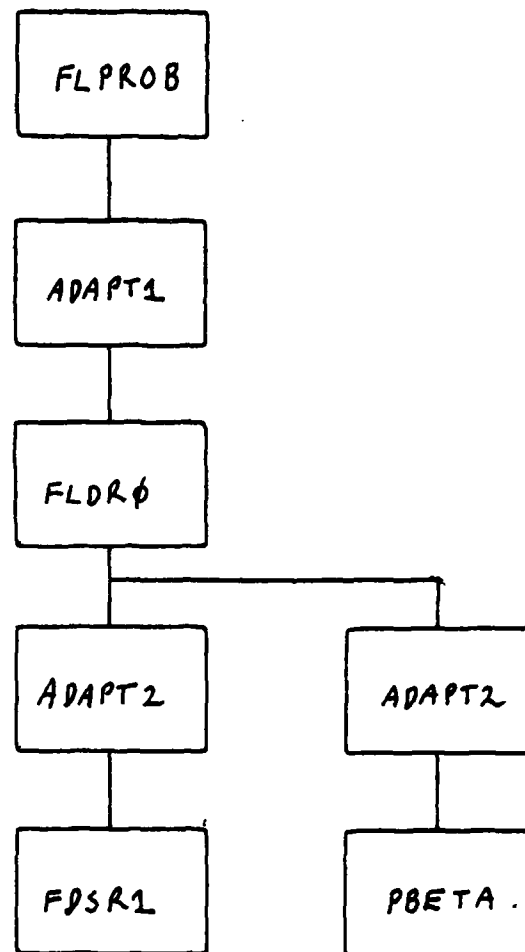


Figure 5.29. Function sub-program hierarchy for the evaluation of $p_R(R | F > t)$.

FUNCTION FDS21(ARG)

Function

FDS21 evaluates the integrand of the expression for $f_R(t, n_0)$, i.e. a function of the form

$$f(\beta) = \psi(\beta) \psi_2(R/\psi(\beta)) H(R/\psi(\beta), \beta, n_0, t) / \psi(\beta). \quad (5.248)$$

Parameter List

ARG: Value of β for which the integrand is required.

Operation

The evaluation of the integrand is straightforward and requires no comment other than the fact that the value of $\psi(\beta)$ is checked to ensure that it is not zero.

Error Message

- (1) 'ERROR IN FDS21 ... PSI IS ZERO': the value of β results in a zero value for ψ . Further operation of FDS21 is prevented.

COMMON Variables Changed

BETV: Current value of β .

PSIVAL: Current value of ψ .

FUNCTION FLDR0(ARG,ERROR1)

Function

FLDR0 evaluates the term, $f_R(R,t,n_0)[p_{a_0}(a_0)]$ used to evaluate $p_R(R|\underline{F}>t)$.

Parameter List

ARG: Value of initial crack length, a_0 , for which the value of the function is required.

ERROR1: Error estimate returned to the calling code and representing the effects of any integrations used in the evaluation of the function.

Operation

The code in FLDR0 controls the logic between the control points FLDR0 and FDSR1 in Figure 5.28 and is divided into two sections. The first section evaluates the contribution from cracked structures. The second section accounts for uncracked structures when appropriate.

(1) Cracked Structures

Full Model

For the full model which accounts for variations in α and \bar{x}_1 , the contribution from cracked structures consists of an integral term which is evaluated using ADAPT2 to integrate FDSR1 over β in the usual way.

The upper limit of integration is given by

$$\begin{aligned} B2 &= \bar{t}_f^*(R) \\ &= \min(\bar{t}_f^*(r_{min}), \psi^{-1}(r/\alpha_{max})\} \\ &= \text{AMIN1}(B2, \text{BRHIGH}), \end{aligned} \quad (5.249)$$

where β has been previously evaluated by RETALI (which returns $\beta = \bar{\epsilon}_2^0(\gamma_{min})$) and α has been evaluated by the calling code and contains the value of $\psi^{-1}(\rho/\alpha_{max})$.

Constant α , Varying β

If α is constant the term for uncracked structures involves no integrations. Given ρ , β is effectively determined as $\psi^{-1}(\rho/\bar{\epsilon}_0)$, where $\bar{\epsilon}_0$ is stored in ALPV. The required expression contains the factor

$$f(\beta) = \rho_\beta(\beta) / [r_0 \psi'(\beta)] \quad (5.250)$$

which can be evaluated by the code calling FLDR0 and stored in CONT1. To complete the evaluation, FLDR0 needs only to include the loss factor if required so that the contribution from cracked structures can be written as

$$FLDR0(\alpha) = CONT1 \exp(-RNSV * GVAL(ALPV, BETV)) \quad \dots (5.251)$$

Constant β , Varying α

If $\bar{\epsilon}_1$ is constant, β is fixed (for given n_0 and t) at $n_0 + t$. If $\beta < \bar{\epsilon}_1$, then FLDR0 consists of a contribution from uncracked structures only. This 'virgin' term is evaluated for this class of model by the first section of code in FLDR0 and is given by

$$f_R(R, t, n_0) = \rho_\alpha(R) \exp(-r_1(R)t) \quad (5.252)$$

$$= PALPHA(R) \exp(-RNSV * RLOADV)$$

where RLOADV has been evaluated by the calling code.

If $\bar{\epsilon}_1 \leq \beta \leq \bar{\epsilon}_2$ the only contribution to FLDR0 is from cracked structures and

$$f_R(R, t, n_0) = \rho_\alpha(\alpha) H(\alpha, n_0 + t, n_0, t) / \psi(n_0 + t)$$

$$= \text{PALPHA}(\text{ALPV})/\text{PSIVAL}$$

$$* \text{EXP1}(-\text{RNSV} * \text{GVAL}(\text{ALPV}, \text{BETV}))$$

... (5.253)

$$\text{where } \alpha = \text{ALPV} = R/\psi(\beta) = \text{RVAL}/\text{PSIVAL}$$

Constant α and β

An evaluation for a model with α and β constant occurs only when $\beta = t + n_0 = \bar{\psi}^{-1}(R/\bar{R}_0)$. If $\beta < \bar{t}_1$,

For $\bar{t}_1 \leq \beta < \bar{t}_2$,

$$f_R(R, t, n_0) = H(\bar{R}_0, \beta, n_0, t) / (\bar{R}_0 \psi'(\beta)) \cdot da_0/dn_0$$

$$= 1/\text{ALPV}/\text{PSIDEV}(\text{BETV}) * \text{CHKDEV}(\text{RNO})$$

$$* \text{EXP1}(-\text{RNSV} * \text{GVAL}(\text{ALPV}, \text{BETV}))$$

... (5.254)

(ii) Uncracked Structures

The second section of FLDR0 is executed for models which do not have \bar{t}_1 constant. Each time RNSV changes, $P_\beta(\bar{t}_1)$ is evaluated using ADAPT2 to integrate PBETA over β . For a model with α constant, the term is completed by evaluating the loss factor, $H(R, \beta, n_0, t)$. For models with α varying, the density function $p_\alpha(R)$ is also included.

5.8.2. Failure density for strength

The failure density for strength is given by equation (3.88), i.e.,

$$p_{\underline{R}}(h, t) = \int_{a(n_{R1})}^{a(n_{R2})} [f_{\underline{R}}(R, t, n_0) r_2(R) + f_{\underline{R}, f2}(R, t, n_0) \delta(R, R_{\min}) p_{ff1}(t, n_0)] p_{a_0}(a_0) da_0 / (F_{\underline{S}}(t) r(t))$$

.... (5.255).

The only new function in (2.55) is $f_{\underline{R}, f2}(R, t, n_0)$ which is defined by equations (3.100), (3.123), (3.143) and (3.166). These equations, which are zero for the models in which α is constant are summarised in Table 5.20.

The logic associated with the evaluation of $f_{\underline{R}, f2}(R, t, n_0)$ is shown in Figure 5.30. The function sub-program hierarchy is given in Figure 5.31.

The term $p_{ff1}(t, n_0)$ can be handled separately by taking it out of equation (5.255) thus,

$$T_f = \int_{a(n_{R1})}^{a(n_{R2})} p_{ff1}(t, n_0) p_{a_0}(a_0) da_0 / P_{\underline{S}}(t) r(t), \quad (5.256)$$

which is to be evaluated for $R = R_{\min}$. The term, $T_f \cdot (P_{\underline{S}}(t) r(t))$ can be evaluated using the code used to evaluate $r_f(t) \cdot P_{\underline{S}}(t)$, provided the contribution made by $p_{ff2}(t, n_0)$ is omitted from the calculation.

Failure Density for Strength

$$p_R(R|t) = \int_{a(n_{R1})}^{a(n_{R2})} f_R(R, t, n_0) r_2(R) + \gamma(R, R_{\min}) p_{ff1}(t, n_0) + \\ + f_{R,f2}(R, t, n_0) p_{a_0}(a_0) da_0 / (P_S(t) r(t))$$

Class	$f_{R,f2}(R, t, n_0)$
Full $\alpha = R/\psi(\tilde{t}_f)$ $\beta = \tilde{t}_f$	$\frac{H_S(R - R^*(\tilde{t}_1, \tilde{t}_f)) p_\theta(\tilde{t}_f) (\tilde{t}_f - n_0) p_\alpha(\alpha) \delta(\tilde{t}_f, \tilde{t}_f^*(R_{\min}))}{t \psi(\tilde{t}_f)} \\ \times H(\alpha, \tilde{t}_f, n_0, t)$
α Const.	0
λ_1 Const. $\alpha = R/\psi(\tilde{t}_f)$ $\beta = \tilde{t}_f$ $n_0 = \tilde{t}_f - t$	$H_S(R - R^*(\tilde{t}_1, \tilde{t}_f)) p_\alpha(\alpha) H(\alpha, \tilde{t}_f, \tilde{t}_f - t, t) / \psi(\tilde{t}_f) \\ \times \delta(\tilde{t}_f, \tilde{t}_f^*(R_{\min}))$

Table 5.20 Summary of expressions for $f_{R,f2}(R, t, n_0)$.

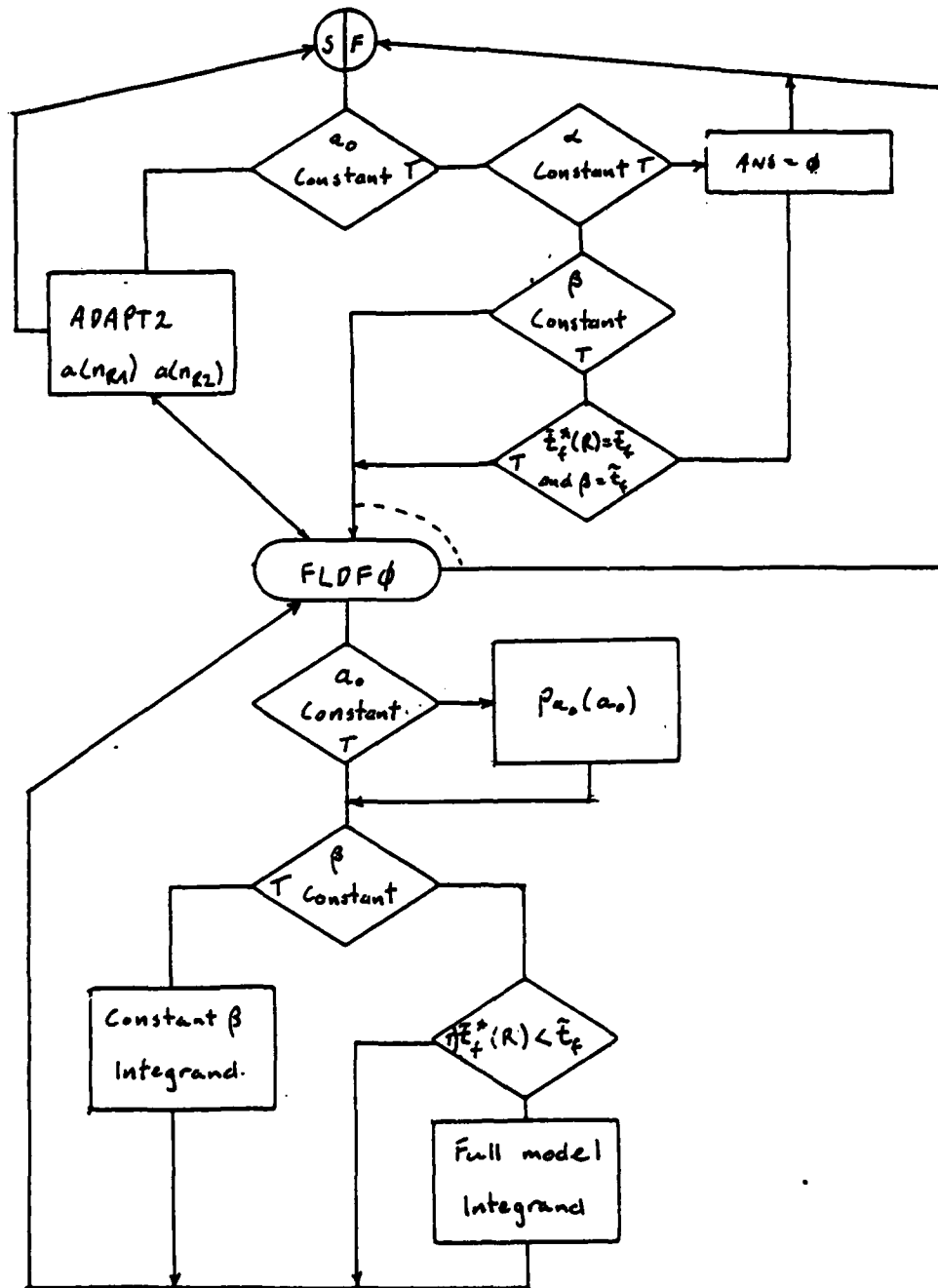


Figure 5.30. Logic associated with the evaluation of the $f_{R,f2}(R,t,n_0)$ contribution to $p_R(R/t)$.

464

467

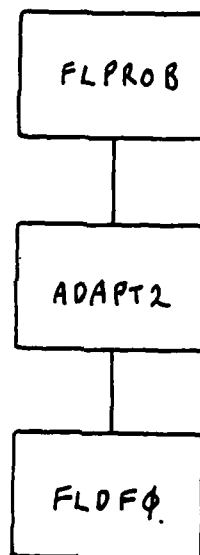


Figure 5.31. Function sub-program hierarchy associated with the evaluation of the $f_{R,f2}(R,t,n_0)$ contribution to $p_{\underline{R}}(R|t)$.

The variable NTERM provides the facility of 'turning off' the $p_{ff2}(t, n_0)$ contribution in FRFO. If NTERM is set to 1, the $p_{ff2}(t, n_0)$ logic in Figure 5.26 is bypassed.

For the case $R = R_{\min}$, the evaluation of the term $T_f \cdot P_{\tilde{S}}(t) r(t)$ replaces the term resulting from $f_{R, f2}(R, t, n_0)$. The result of either calculation is stored in the variable TRMRF.

At the completion of a single pass through the DO LOOP that sweeps through the R values (in FLPROB) the variables TERM and TRMRF have the following values;

$$\text{TERM} = \int_{a(n_{R1})}^{a(n_{R2})} f_R(R, t, n_0) p_{a_0}(a_0) da_0, \quad (5.257)$$

$$\text{TRMRF} = \int_{a(n_{R1})}^{a(n_{R2})} f_{R, f2}(R, t, n_0) p_{a_0}(a_0) da_0 \quad (5.258)$$

or if $R = R_{\min}$,

$$\text{TRMRF} = \int_{a(n_{R1})}^{a(n_{R2})} p_{ff1}(t, n_0) p_{a_0}(a_0) da_0 \quad (5.259)$$

so that $p_{\tilde{R}}(R|t)$ is given by

$$P_{\underline{R}}(R|t) = (TERM \cdot r_2(R) + TERMRF) / (P_{\underline{S}}(t)r(t))$$

$$= (TERM \cdot RLOAD(R) + TERMRF) / RSLTDS(J) \quad (5.260)$$

where RSLTDS(J) contains the value of $P_{\underline{S}}(t)r(t)$ which was computed during the construction of the time sequence of reliability functions.

AD A145 685

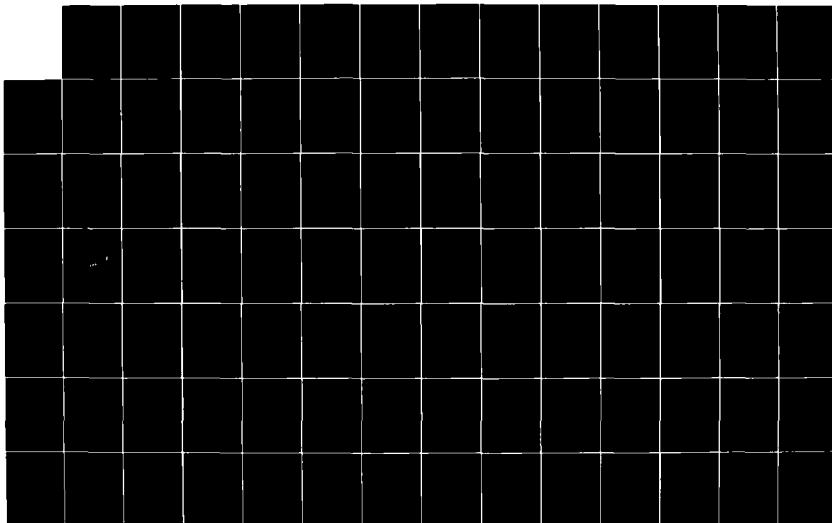
NERF - A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUN. (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARI/STRUC-397

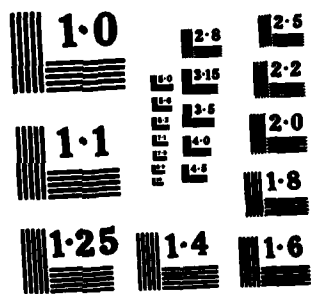
6/7

UNCLASSIFIED

F/G 9/2

NI





FUNCTION FLDF0(ARG)

Function

FLDF0 evaluates the function $p_{a_0}(a_0)f_{R,f2}(t,n_0)$ which is used to compute the contribution to $n_0(t)$ made by $r_f(t)$ type failures.

Parameter List

ARG: Value of initial crack length a_0 for which the evaluation is required.

Operation

FLDF0 is used for only two classes of model. In each case the function contains the factor $p_a(\alpha)/\psi(\bar{t}_f)$ which is evaluated by the calling code and stored in CONT2.

Prior to the evaluation of the function, $\bar{t}_f^*(t)$ is evaluated and if less than \bar{t}_f , the FLDF0 is returned with a zero value. Otherwise either

$$f_{R,f2}(t,n_0) = \text{CONT2} * \text{EXP1}(-\text{RNSV} * \text{GVAL}(\text{ALPW}, \text{PWT})) \quad \dots (5.261)$$

if β is constant, or

$$f_{R,f2}(t,n_0) = \text{CONT2} * \text{PASTA}(\text{PWT}) * (\text{PWT} - \text{RVT}) / \text{RNSV} \\ \text{EXP1}(-\text{RNSV} * \text{GVAL}(\text{ALPW}, \text{RVT})) \quad \dots (5.262)$$

if β is not constant.

COMMON Variables Changed

RVT: Current value of n_0 .

5.9. Basic Communications, Input-Output Procedures and Mathematical Functions

The computer code described to this stage has been concerned primarily with the evaluation of the reliability functions or the operation of the various numerical algorithms. There are several subroutines and function sub-programs that provide basic communications facilities and other essential requirements for the program's operation. These subroutines are described in this Section.

5.9.1. Communications

NERF can operate in two modes, 'interactive' or 'batch'. In the interactive mode, a dialogue appears on the remote terminal or visual display unit (VDU). This dialogue contains prompts, error messages and requests for user responses necessary to select optional facilities as the run proceeds. In the 'batch' mode, these messages are suppressed from transmission to the VDU but most appear on the secondary output file.

All prompts and messages pass through the subroutine PROMPT which can detect the mode of operation and process the messages accordingly. Requests for user responses also pass through basic communications subroutines (INTEST, INTGIN, REALIN and TXTIN) which use PROMPT to channel the messages as appropriate as well as providing a standard format for the requests and subsequent responses.

The greater bulk of the code in NERF therefore contains no direct input-output statements (such as READ, WRITE, TYPE and ACCEPT) and is independent of the communications facilities provided by the computer system on which NERF is installed. Changes to those facilities can be accommodated by making a few changes to the communications routines rather than changing the whole code.

The facilities provided by the communications routines are described below.

(i) Text string output (PROMPT)

A prompt or message consists of a text string containing a specified number of characters. To transmit a message to the terminal and secondary output file, a call to PROMPT is made, e.g.

```
CALL PROMPT('THIS IS A MESSAGE',17).
```

(ii) Echo of user response (ECHO)

A response entered on the terminal is read by the code in the routines REALIN and TXTIN (the only subroutines in NERF that accept user responses) as a text string. ECHO provides a facility for transmitting an echo of the response to the terminal and to the secondary output file.

In the current installation on the DEC SYSTEM-10 at ARL there is no need to echo the response on the terminal and the response is transmitted to the secondary file only. This version of ECHO does, however, interact with PROMPT by adjusting the first character, which is always a 'control' character, sent by PROMPT so that the effect of a linefeed echoed by the system following a user response is suppressed.

(iii) Text string input (TXTIN)

Subroutine TXTIN is intended to provide the facility for prompting and accepting a user response consisting of a string of characters that will be interpreted as text. The call is of the form,

```
CALL TXTIN('FILE NAME?',10,ANS,1)
```

where the text string containing the prompt and the number of characters in that string constitute the first two parameters in the call. The user's response will be returned in the variable ANS. The last parameter indicates the number of words allocated for storage of the response. .

(iv) Real number input (REALIN)

REALIN provides a similar facility to that described above when the user's response is to be interpreted as one or more numbers.

All processing of the response to form the numbers is handled by REALIN which contains provision for the construction of lists of numbers. (See documentation below.)

(v) Integer number input (INTGIN)

INTGIN provides a similar facility to that of REALIN but for integer numbers. INTGIN, in fact, calls REALIN so that user responses for all numbers are the same and are converted, when appropriate, by the code in INTGIN.

(vi) Logical input (INTEST)

Often, the user is asked to select between two alternative options and is asked to reply 'yes' or 'no' to a suitable question. INTEST provides this facility. For example a statement of the form,

```
IF(INTEST('DO THIS?',10)) GO TO 10
```

will result in the prompt 'DO THIS?' appearing on the screen. If the user replies 'YES' or 'Y' a jump to statement 10 will be executed. Otherwise control will pass to the next statement.

Further details of the operation of the communications routines are given below.

475

SUBROUTINE ECHO(TXT,NCH)

Function

ECHO transmits a text string to the secondary output file only. It is intended that ECHO is used to transmit keyboard entries only. However there is no coded test to prevent alternative use.

Parameter List

TXT: Array containing the text string to be transmitted.

NCH: Number of characters in the text string.

Operaton

ECHO operates in a similar manner to PROMPT in that the last word in TXT is modified so that any characters beyond NCH are replaced by blanks before transmission. The string is preceded by a '*' to identify a keyboard response on the secondary output file.

The first character in FCHR (in common block BATCH) is set to '+' so that the next string transmitted by PROMPT (see below) will supress transmission of a new line.

LOGICAL FUNCTION INTEST(A,NCH)

Function

INTEST returns a logical value of true or false, depending on whether the operator has entered 'yes' or 'no' on the keyboard from which NERF is being operated.

Parameter List

A: Array containing a text string which will invite the operator to make the required response.

NCH: The number of characters in the string stored in A.

Function

INTEST uses TXTIN to input the operator's response to the prompt stored in A. This response is checked and INTEST set true if 'YES', 'Y' or 'T' has been entered. INTEST is set false otherwise. Note the response must be upper case. The response 'T' is permitted to ensure compatibility with previous versions of NERF which requested a true or false answer directly.

Prompts

(1) Prompt stored in A.

SUBROUTINE INTGIN(TXT,NCH,IVALS,NUM)

Function

INTGIN controls the input of integer constants from the keyboard of the terminal from which NERF is being operated.

Parameter List

TXT: Array containing a text string which will be used to prompt the operator to enter data on the keyboard.

NCH: Number of characters in the text string stored in TXT.

IVALS: Integer array into which the constants will be transferred by INTGIN and returned to the calling code.

NUM: >0; The number of integers to be input. The number is fixed and not subject to variation by the operator and all the integers will be entered on the one line.

<0; The number of integers is not fixed and a series of prompts will instruct the operator accordingly.

=0; Same as above with the maximum number of constants set by default to 100.

Operation

INTGIN operates by calling REALIN to input a set of real constants which are then converted to integers. This means that data entry is free of format restrictions and real numbers (e.g. 1e4 for 1000) can be entered if convenient.

For further details of the modes of data entry refer to the description of subroutine REALIN.

SUBROUTINE PROMPT(TXT,NCH)

Function

PROMPT handles the transmission of text strings to the terminal from which NERF is being operated. The text is also transmitted to the secondary output file which contains a detailed account of the operation of the NERF program and error messages.

Parameter List

TXT: Array containing the required text string in A5 format.

NCH: >0; The number of characters in the text string.

=0; A blank line is transmitted.

<0; NCH defines the number of characters in the string which is preceded during transmission by a blank line.

Operation

This subroutine is very dependent on the computer system on which NERF is implemented. The current version is compatible with the DEC-10 computer operating at ARL during 1982.

Transmission to the terminal is controlled by a TYPE statement; transmission to the secondary output file by a WRITE to logical unit number 4. The number of characters per word on the DEC-10 is 5 which is the value assigned in the routine to NCHWD. However the code does make the assumption in other sections that each word has 5 characters and implementation on another machine with different word length would require careful reconsideration of the code.

Prompt modifies the last word in TXT so that any characters beyond NCH are replaced by blanks. This simplifies output to the terminal, but does overwrite information in TXT.

Transmission to the terminal is preceded by a control character which may be used to effect line spacing control if required. This character is stored as the first character in FCHR in the common block BATCH. Currently this character is used to suppress a new line following entry of data on the keyboard of the terminal. After such entry, the operating system sends a new line to the terminal so that transmission of a normal text string preceded by a 'blank' would result in a blank line appearing on the terminal between the echo of the key board entry and the text string. The insertion of a plus sign in FCHR immediately after a keyboard entry will suppress this blank line, (See SUBROUTINE ECHO).

If the BATCH switch (in common block BATCH) is on, transmission to the terminal is suppressed and the text string is transmitted to the secondary output file only.

SUBROUTINE REALIN(TXT, NUM, VAR, NVAR)

Function

REALIN controls the input of real constants from the keyboard of the terminal from which NERF is being operated.

Parameter List

TXT: Array containing a text string which will be used to prompt the operator to enter the data on the keyboard.

NUM: The number of characters in the text string stored in TXT.

VAR: Real array into which the constants will be stored and transmitted back to the code calling REALIN.

NVAR: >0; The number of constants to be input. The number is fixed and not subject to variation by the operator and all the constants will be entered on the one line. It is envisaged that this mode of operation of REALIN will be used for short (less than 5 say) lists of constants.

<0; The number of constants to be input is not set by the calling code and REALIN will execute a series of prompts to instruct the operator accordingly. The maximum number of constants in this case is -NVAR (usually determined by the space available in VAR).

=0; Same as above with the maximum number of constants set by default to 100.

Operation

REALIN is a subroutine that is dependent on the characteristics of the communications and operating system of the computer on which NERF is installed. The current version is compatible with the DEC-10 operating at ARL in 1982.

For example, the format LOG for data entry allows free format to be used. On other systems the free format facility may have to be invoked some other way.

REALIN operates in two modes, depending on the sign of NVAR. The first mode is appropriate if a fixed list of constants is to be input. The second is used to input a list of constant, the number of which is, itself, part of the information requested by the calling code.

(i) Fixed list mode

The fixed mode is straightforward in operation. The line of text entered on the keyboard is decoded and the results transmitted to VAR. Note that no check is made on the number of constants actually entered on the keyboard. It is assumed that the prompt is sufficiently informative and that the list is short enough to fit on a single line.

(ii) Variable list mode

Realin commences the variable list mode by transmitting the prompt stored in TXT and then inviting the operator to 'enter values 5 at a time - terminate with 0'. If this method is appropriate, the operator enters the value of the constants, 5 to a line. A zero value for a constant is interpreted as indicating the end of the list. NVAR is set to the number of constants input and the results are returned in VAR.

Note that if the number zero is a legitimate value for one of the constants the operator must enter a small number (say 1E-30) to represent that value.

Alternatively, the operator may enter a null line (corresponding to a zero for the first constant. In response, REALIN requests that the 'limits and number of intervals' of a sequence of numbers with fixed spacing be specified.

In either method for entering a variable list, the operator can instruct REALIN to return an empty list by entering 'none' on the keyboard.

(iii) General features

After each line of keyboard entry, ECHO is called to record the entry on the secondary output file.

Each line is checked to ensure that the first character is a legal character for a numerical constant. This traps the likely error that the operator has entered a text string by mistake. Although not a conclusive test for data entry errors, this test has been found to be successful in trapping most keyboard entry errors to date. If an error is detected, the operator is invited to 'enter again'.

Prompts

- (i) Prompt stored in TXT.
- (ii) 'ENTER VALUES 5 AT a TIME - TERMINATE WITH 0'; Initial prompt for variable list mode.
- (iii) '5 VALUES ACCEPTED'; Indication that a line of data has been accepted from the keyboard and more is expected.
- (iv) 'LIST TERMINATED'; In variable list mode, the current input action has been terminated. Normally this occurs following the entry of a 'zero' constant on the keyboard.

Error Messages

- (i) 'ENTER AGAIN'; The first character entered was not legal for the representation of numeric data. The operator must enter the line of data again.
- (ii) 'LIMIT EXCEEDED'; In variable list mode, the operator has entered more constants than the space in the calling code can store. REALIN terminates the input operation and returns the data received to the point of overflow to the calling code.

483

483.

SUBROUTINE TXTIN(TXT,NUM,VAR,NVAR)

Function

TXTIN controls the input of a text string from the keyboard from which NERF is being operated.

Parameter List

TXT: Array containing a text string that is to be transmitted to the terminal to prompt the operator to enter the required text.

NCH: Number of characters in the string stored in TXT.

VAR: Array into which the text is to be input and then returned to the calling code.

NVAR: Number of words in VAR available for the storage of the text.

Operation

TXTIN executes, in a straightforward manner the transmission of the prompt in TXT to the terminal, the decoding of the operator's response into VAR and the transmission of the response to the secondary output file via ECHO.

Prompt

(1) Prompt stored in TXT.

5.9.2. Two-dimensional data array output

There are several requirements in NERF for the output of two-dimensional arrays of numerical data. The subroutines ARROUT and IRRROUT extend the basic communications routines to provide a suitable facility.

The two-dimensional array and its dimensions are passed to the appropriate subroutine (ARROUT for real data and IRRROUT for integer data) which then controls the formatting and eventual output to the secondary output file.

As in the case of the basic communications routines, the provision of these two subroutines frees the remainder of the code of the direct output statements required to write a two-dimensional array to the output files. Moreover, because the output is controlled by a single section of code, changes to the format and/or output destinations are easily made.

485

485

SUBROUTINE IRROUT(X,M,N)

Function

IRROUT controls the printing of the data stored in the two dimensional integer array X into the secondary output file.

Parameter List

X: Integer array containing the data to be printed.

M: First dimension of X.

N: Second dimension of X.

Operation

The data in X is printed in groups of 10 columns, (a column corresponding to the first index being constant). Each row and column is labelled with the appropriate index.

The secondary output file is assumed to be logical unit number 4.

SUBROUTINE ARROUT(X,M,N)

Function

ARROUT controls the printing of the data stored in the two-dimensional real array X into the secondary output file.

Parameter List

X: Real array containing the data to be printed.

M: First dimension of X.

N: Second dimension of X.

Operation

The data in X is printed in groups of 10 columns (corresponding to constant first index). Each column and row is labelled with the appropriate index.

The secondary output file is assumed to be logical unit number 4.

5.9.3. Function file input

Some of the functions used to generate the reliability functions are defined by sequences of ordered pairs of argument and function values. The sequences of ordered pairs are stored in disk files called 'function files' and are read by the subroutine READFN described in this Section.

The output produced by NERF is written as a function file containing several functions, each defined for the same set of argument values. These files are compatible with the format used by READFN and can be read by the subroutine READMF used by NERPLT.

The data preparation program NERPRE contains a subroutine called FCNFIL which enables the user to create function files interactively from the terminal. The files produced by FCNFIL are directly compatible with the requirements of READFN.

SUBROUTINE READFN(X,F,NUM,FNAME)

Function

The subroutine READFN opens, reads and closes a file containing a sequence of ordered pairs defining a function. Such files are referred to in the NERF documentation as 'function files'.

Parameter List

X: Array into which the sequence of argument values will be placed and returned to the calling code.

F: Array into which the sequence of function values will be placed and returned to the calling code.

NUM: Number of ordered pairs defining the function, (returned to the calling code.)

FNAME: File name - in the form of a 5 character string determining the root for the name of the file containing the sequence of ordered pairs. All function files have the extension '.FCN' .

Operation

The file name is formed from the 5 characters in FNAME and the standard extension, .FCN. For example, if the name,

'DATA1' is passed in via FNAME, READFN will look for a file called 'DATA1.FCN'.

The file is opened and read according to the format specification in Table 5.21. The end of the data is defined by the physical end of the file.

Note that the parameter ICOL, defining the number of columns is read from the file. READFN reads only a single function. A companion subroutine in NERPLT (called READMF) can input several functions from the one file. All the functions are assumed to be defined for the same set of argument values and are hence stored in 'columns' in the file. READFN is compatible with READMF in that it reads the first column of of a multi-function file.

After reading the function, READFN echoes the data on the primary output file. Note that this is one of the few subroutines in NERF which writes directly to one of the output files.

The format specification in Table 5.21 is met by READFN, READMF (in NERPLT), the output statements in PROGRAM NERF and . OUTPUT and the subroutine FCNFIL in NERPRE which prepares function files.

Record	Format	Variable	Significance
1	12A5	} TITLE	3 lines of 60 characters identifying data
2	12A5		
3	12A5		
4	I3	ICOL	Number of columns or functions
5	22(2X,2A5)	HEAD	Headings identifying up to 10 functions
6	11G	X(1), F(1), F2(1), etc.	1st argument value 1st function ₁ value 1st function ₂ value etc.
7	11G	X(2), F(2), F2(2), etc.	2nd argument value 2nd function ₁ value 2nd function ₂ value etc.

Table 5.21. Format specification for the function files.
The table indicates the format for the most
general 'multi function' file. READFN reads
the first two numbers (X,F) in each record.

5.9.4. Run time monitoring and program termination

The subroutine EXTIME provides a run-time monitoring facility which can be used to limit the 'cost' of a given run by NERF. Because the integration algorithms are adaptive, the run-time for a given calculation can not be predicted. In particular, considering the possibility that the integration algorithms have a finite (but small) potential for failure which manifests as a marked increase in the number of function evaluations to effect a given integration, some control of the run time is required.

EXTIME determines the current run time: if this run-time is greater than a pre-determined limit the program operation is terminated by transferring control to the subroutine FINISH. The estimation of a program's execution or run time is not standard within the FORTRAN language. The implementation for the DEC SYSTEM-10 is described below - the routine localises the calls to the system supplied timing routines for easy modification should NERF be installed on another computer.

EXTIME calls CFNEW to ensure that the next run can start from where the terminated run left off.

Program termination is executed by the subroutine FINISH which computes the total run time before finally terminating the run. The main objective in providing this routine was, again, to localise the system dependent timing and job termination code.

SUBROUTINE EXTIME

Function

EXTIME determines the current run time of the computer program and terminates operations if a preset limit is exceeded.

Operation

The allowable run time, in minutes is stored in RTIME in the common block RTIME. Access to the clock is installation dependent. On the DEC-10 at ARL a system subroutine called TIMES returns the current job time, (not run time), in milliseconds. Thus when NERF starts, TIMES is called (by the main program, NERF) to obtain a reference time which is stored in IT1, again in the common block RTIME. This value is used to convert the current job time to an elapsed time.

If the preset time is exceeded, control is passed initially to CFNEW to update the data control file and then to FINISH to terminate the operation of the program.

Prompt

- (1) 'TIME LIMIT EXCEEDED'; This prompt identifies the reason for the termination of the operation of the program.

493

493

SUBROUTINE FINISH

Function

FINISH terminates the operation of NERF.

Operation

The run execution time is computed, (see EXTIME for details) and an informative message is transmitted to the terminal and placed on the relevant output files.

Prompt

- (1) 'RUN ***** TERMINATED'; (The run identification is inserted in the prompt.)

5.9.5. Mathematical functions

Generally NERF relies on the mathematical functions provided as part of any FORTRAN operating system. The exceptions are detailed below.

(i) Exponential function

The function EXP1 provides a replacement for the usual function installed in the operating system, and traps large values of the argument - generating an appropriate warning message when it does.

(ii) Logarithm function

The function ALOG1 intercepts a zero value for the argument and returns a value of -80.0 to the calling code. This allows the calling code to cope with a possible error condition.

(iii) Gamma function

The evaluation of the gamma density function (5.53) uses the function S14AAF provided by the NAG library of scientific subroutines. This is the only mathematical function not provided by a standard FORTRAN system.

495

47

FUNCTION ALOG1(ARG)

Function

ALOG1 is a substitute for the natural logarithmic function provided by the system software. This replacement returns the value -80.0 if the argument is zero.

Parameter List

ARG: Argument for the natural logarithm function.

Operation

ALOG1 uses the system function ALOG to evaluate natural logarithm for values of ARG greater than or equal to 10^{-30} . Otherwise the value -80.0 is returned.

FUNCTION EXP1(ARG)

Function

EXP1 is a replacement for the exponential function provided by the system software. The replacement function provides an error message if the argument exceeds 80.0.

Parameter List

(ARG): Argument for the exponential function.

Operation

If ARG exceeds 80.0, an error message is transmitted to the terminal and the secondary output file. No further action is taken so that EXP1 returns the evaluation made during the previous call.

If $ARG \leq 0$, the value 0.0 is returned.

Error Messages

- (1) 'EXP1 ... ARG OUT OF BOUNDS'; The argument of EXP1 exceeds 80.0.

5.10. Graphics Support

As described in Section 5.2.6., the program NERF provides for the construction of three types of graphical representations of the processes involved in the evaluation of the reliability functions, viz, outer integrand plots, integrand - function evaluation maps and loss factor maps. The post processor NERPLT provides facilities for plotting the functions produced by NERF.

The documentation in this Section concentrates on the facilities provided by NERF itself. However, the program NERPLT is briefly described in Section 5.10.5. for completeness. NERPLT makes use of several of the subroutines contained in the NERF program and presented in this document.

The library of graphics subroutines forms a separate entity within the NERF program. Graphics facilities can be removed by loading a dummy set of subroutines in place of this library. Because the associated prompts are contained within the graphics library, the program will then behave as if there were no graphics facilities.

The description of the graphics subroutines follows the top-down sequence of presenting the groups of subroutines associated with particular representations followed by the group of subroutines that provide more basic facilities.

Before embarking on this description, the following general comments are in order.

- (i) All the graphical output produced by a single run of the NERF program is transmitted to a file called FNAME.PLT on logical unit number 5. FNAME is the 5 character run identification.
- (ii) Graphics operations are controlled by the logical switch PLTINT. At the commencement of an interactive run the operator is asked if integrand plots are required. An affirmative reply switches PLTINT on and the graphical facilities are activated. Otherwise graphical facilities are not active for the whole run.
- (iii) The graphics subroutines rely on the plotting library installed on the DEC - SYSTEM 10 at ARL. Although there is a general tendency for those subroutines to be called more frequently from the more basic graphics routines, the code has not been structured to the extent that only the basic routines rely on the particular plotting subroutines provided by the computer installation on which NERF is operating.

*The option to create plots is offered only if there is a single evaluation time for the run.

5.10.1. Outer integrand plots

The construction of an outer integrand plot consists of two phases. During the first phase, the integrand evaluations are stored as the integrand is evaluated. The second phase commences after the integral has been evaluated and contains those operations necessary to construct the plot.

Three high level subroutines are involved. Subroutine PLTSET initialises the storage areas used to store the integrand evaluations. PLTSTR stores the pairs of integration variable and integrand values as the integration proceeds and PLTOUT plots the stored values as a graph having the general form of the example given in Figure 5.7.

It is up to the calling code to ensure that the integrand values are stored. This is why the various integrand functions in NERF have code which detects whether the particular integrand function is the outermost integrand and, if so, calls PLTSTR to store the evaluation.

The three subroutines described in this section perform the general operations described above. The details of the actual plots created are dictated to a large extent by the subroutines described in Section 5.10.4.

SUBROUTINE PLTOUT(TITLE,NCH,LABX,NX,LABY,NY)

Function

Subroutine PLTOUT constructs a two-dimensional plot of data pairs that have been stored during a series of previous calls to PLTSTR.

Parameter List

TITLE: Text string containing a heading for the plot.

NCH: The number of characters in the heading.

LABX: Text string containing a label for the X axis, (representing the argument).

NX: The number of characters in LABX. (Only the first 5 characters are used.)

LABY: Text string containing a label for the Y axis, (representing the function).

NY: The number of characters in LABY. (Only the first 5 characters are considered.)

Operation

PLTOUT uses SETGRF to initialise a suitable plot space and then sweeps through the data pairs stored in FARG and FVAL, constructing a user selected symbol to represent each point. To do this, PLTOUT determines the ranges of the argument and function values which are then used as a basis for an interactive dialogue to determine the limits for the graph and any other information required by SETGRF. Once the plot space has been initialised each data pair is scaled and vignnetted, if necessary, before the subroutine POINT is called to construct the symbol.

Error Messages

- (i) 'NOTHING TO PLOT': PLTOUT has been called with empty data arrays. No plot is created.
- (ii) 'INSUFFICIENT DATA TO PLOT': PLTOUT has been called with only one data pair in storage. No plot is created.

Prompts

- (i) 'OUTER INTEGRAND PLOT'
'ARGUMENT VALUES ARE BETWEEN **** AND ****'
'XMIN,XMAX,XSTEP'
Suitable limits for the argument together with an appropriate interval for the labelling of the X axis are requested.
- (ii) 'INTEGRAND VALUES ARE BETWEEN **** AND ****'
'YMIN,YMAX,YSTEP'
The limits for the integrand data to be plotted and a suitable interval for the labelling of the Y axis are requested.
- (iii) 'POINT CODE': A number identifying the the type of symbol to be used to represent each data pair is requested. (For code values refer to Table 5.22.)

Storage Conflict

The arrays FARG and FVAL used to store the data pairs occupy space in the GEXP array in COMMON block GCOM.

POINT CODE	SYMBOL
1	Triangle
2	X
3	+
4	Square
5	Point
6	Octagon
ASCII String	First character of string.

TABLE 5.22. Values for 'point code' and the resulting symbols.

(Prepared from DECsystem documentation)

SUBROUTINE PLTSET

Function

PLTSET clears the storage areas used for data pairs that will be ultimately used for the construction of 'outer integrand plots'.

Operation

The clearing operation is effected by simply setting IMAX to zero.

PLTSET also sets the switch PLOTON which flags whether an outer integrand plot is required or not. The flag remains operative until the next call to PLTSET

Prompt

- (1) 'OUTER INTEGRAND PLOT?': Prompt requests a response from the operator to set the switch PLTON. It is assumed that a prompt indicating the integration currently being made has been issued.

SUBROUTINE PLTSTR(ARG,VAL)

Function

Subroutine PLTSTR stores a data pair consisting of an argument and the value of an associated function for later plotting as a series of points on a two-dimensional graph. This construction is used by NERF to construct 'outer integrand plots'.

Parameter List

ARG: Value of the argument for the data pair.

VAL: Value of the function for the data pair.

Operation

The role of PLTSTR is to store the argument and function values in the arrays FARG and FVAL respectively. As each data pair is stored an index, IMAX, is incremented so that it always indicates the total number of data pairs in storage.

Note that if the switch PLOTON is false, no action is taken by PLTSTR.

5.10.2. Integrand - function evaluation maps

Contour maps for the functions $p_{fs}(t, n_0)$ and $P_F(t, n_0)$ are constructed for the full model with n_0 constant if the run is interactive and the operator has responded in the affirmative to the appropriate prompts. The maps are constructed by the subroutine INTPLT which is called from the function FRLTO.

FRLTO is used as the integrand function for both $P_F(t)$ and $r_g(t)$. $P_g(t)$: the switch, RISK, is used to distinguish between the two functions. This logical variable is used by FRLTO to select an appropriate heading for the map: functions are identified here and by the program NERPLT as listed in Table 5.23. The switch also selects the appropriate terms evaluated by the integrand function FALP.

For reference, a typical integrand map is shown in Figure 5.32. The map is constructed by evaluating the integrand at points in (α, β) space which correspond to the intersections of set of lines constructed in the same way as the interpolation mesh used for the loss factor. However in this case a different set of β values is used. The selection of these β values is described below in the documentation for INTPLT.

Looking at the map, the sequence of operations for its construction are as described below.

- (1) The plot space is initialised by the subroutine SETGRF which draws the box around the plot area, annotates the axes and

Heading	Symbol	Significance
LOAD	L	Load
BETA	β	Age
ALPHA	α	Virgin strength
AGE	\bar{t}	Age
LENGTH	a	Crack length
AO	a_0	Initial crack length
EXEC RATE	$\bar{F}_L(R)l_r$	$r_2(R)$
PSI	$\psi(\beta)$	Median strength decay function
STRENGTH	R	Strength
ALPHA DIST	$p_\alpha(\alpha)$	Density for
BETA DIST	$p_\beta(\beta)$	Density for
P(AO)	$p_{a_0}(a_0)$	Density for a_0
F(a0)	$F_{a_0}(a_0)$	Outer integrand for a_0 .
F(BETA)	$F_\beta(\beta)$	Outer integrand for
F(ALPHA)	$F_\alpha(\alpha)$	Outer integrand for
RISK	$r(t)$	Total risk
RISKS	$r_s(t)$	Risk of static fracture
RISKf	$r_f(t)$	Risk of fatigue life exhaustion
RISKV	$r_v(t)$	Virgin risk
MEAN RISK	$r_{\text{mean}}(t)$	Mean risk
PR. FAIL	$P_F(t)$	Probability of failure
PR. SURVIV	$P_S(t)$	Probability of survival
AV. LIFE	$E(\bar{F})$	Expected time of failure
PR. REJECT	$P_{\text{det}}(t)$	Probability of detection
RSLT	$r_s(t) \cdot P_S(t)$	
PF	$P_F(t)$	Probability of failure
FLD	$p_R(R t)$	Density for failing load
FLP	$P_R(R t)$	Distribution of failing load
RSD	$p_R(R \bar{F}>t)$	Density for strength
RSP	$P_R(R \bar{F}>t)$	Distribution for strength.

Table 5.23. Headings used on plots and their meanings.

504

507

UM324 RSLT INTEGRAND

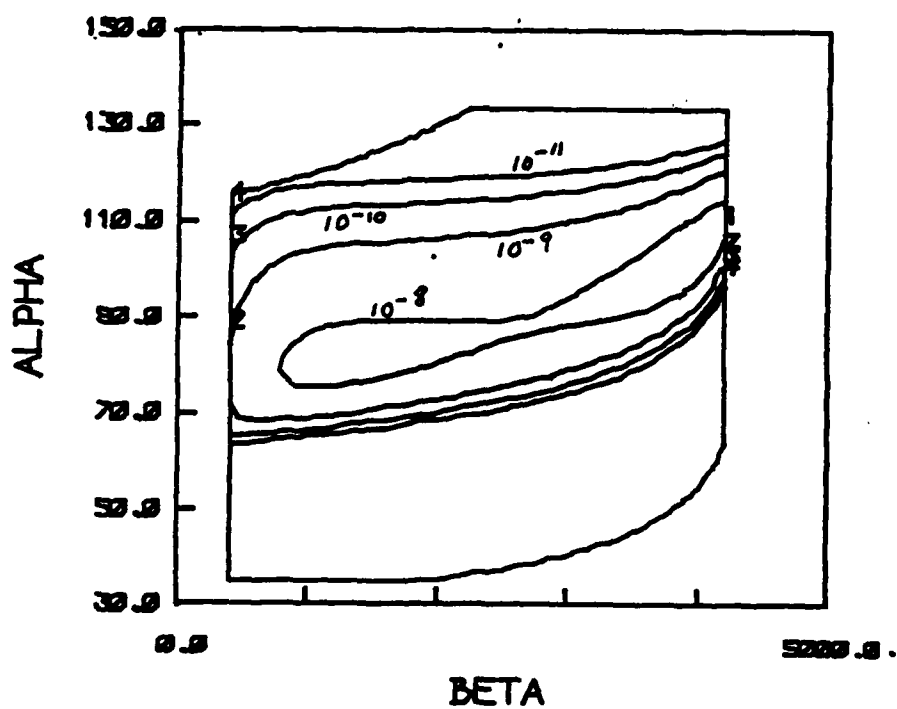


Figure 5.32. Example of an integrand plot. Integrand is $p_{fs}(t, n_0)$ for Example A and $t = 1200$ hrs.

writes the title above the plot space.

- (ii) The (α, β) domain is initialised by the subroutine SETPLT which also draws the domain boundaries.
- (iii) The actual map is constructed by the subroutine ARRPLT which calls a contouring subroutine, CONT. This subroutine constructs the contour lines with reference to the indices of the array used to store the values of the integrand. The mapping between those indices and the (α, β) space is controlled by the subroutine P which eventually plots the points generated by CONT. Subroutine P also labels each contour line by writing an identification number at the start of each contour line.
- (iv) Extra lines indicating inspection boundaries can be drawn on the map as shown, for example, in Figure 5.9. This action is controlled by INTPLT.
- (v) The subroutine INTPLT writes the contour levels beneath the map. (This text was removed from the plot used to produce Figure 5.32 and the contour levels were written beside the lines by hand.)

The various subroutines are not called in the sequence indicated above, but instead, form a hierarchy as indicated in Figure 5.33. Only the subroutine INTPLT is exclusive to the integrand map construction process and is described in this Section.

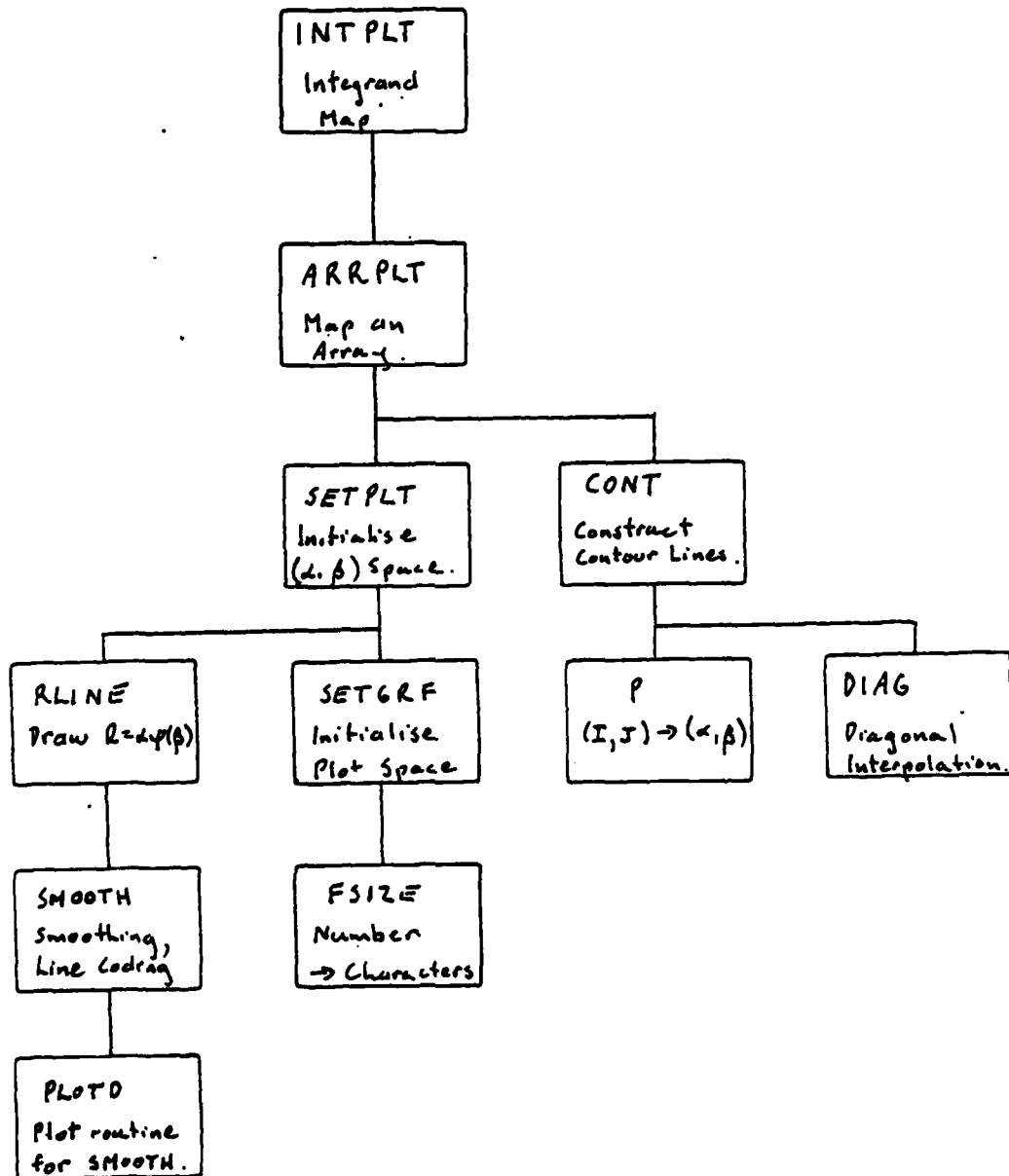


Figure 5.33 Subroutine hierarchy used to construct maps in (α, β) space.

After the map has been constructed, FRLTO continues with the evaluation of the double integral expression defining the function portrayed by the map. During this evaluation the subroutine PLTPNT is called to plot each point at which the integrand is evaluated. A typical result of this action is shown in Figure 5.34.

As described in the documentation for PLTPNT below, the insertion of the evaluation points on the integrand map can be restricted to subdivisions above nominated levels (different levels for α and β) so that the final plot does not become too cluttered.

UM319 RSLT INTEGRAND

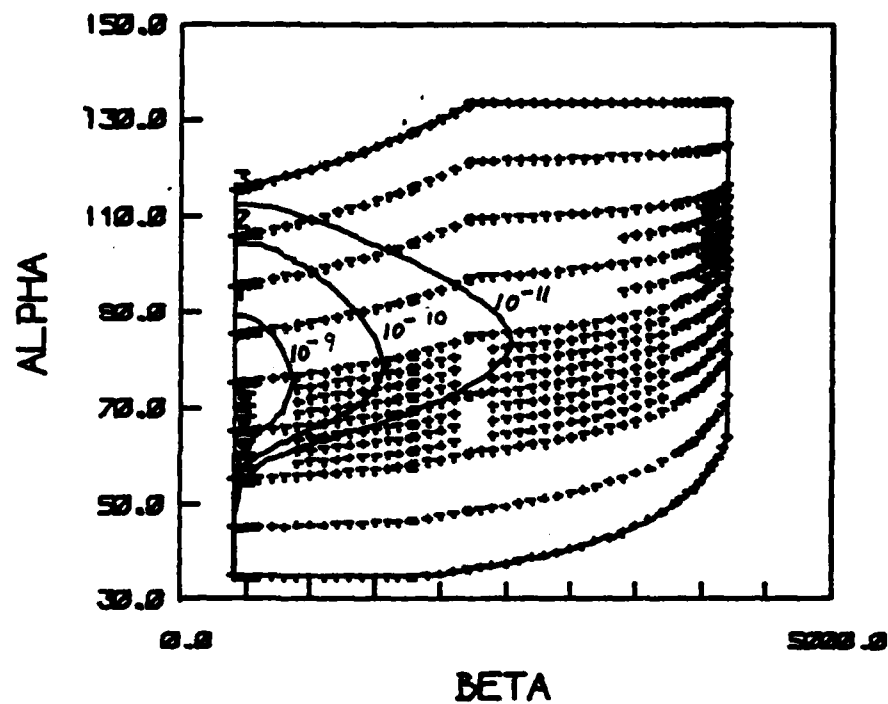


Figure 5.34 Integrand plot with locations
of integrand evaluations indicated.
Example A, $p_{fs}(t, n_0)$, $t = 500$ hrs.

SUBROUTINE INTPLT(TITLE,NCH,B1,B2)

Function

INTPLT controls the construction of a contour map of the integrand of a two-dimensional integration over α and β using the function FALP. The subroutine also controls the construction of representations of the various integration bounds that are applicable.

Parameter List

TITLE: Text string containing a suitable title for the map.

NCH: Number of characters in the title.

B1: Lower β limit for the integration.

B2: Upper β limit for the integration.

Operation

The map is constructed over the same region in α, β space over which the loss factor interpolation table is described. The mesh used to define the point values of the integrand used for the map resembles that used for the loss factor but has more values.

The actual number of additional nodes depends on the available space in the GEXP array, (used to store the point values), and the value of an integer NTYP which sets the typical resolution for the β direction.

The β nodes used to define $\psi(\beta)$ are used as a basis, additional nodes being inserted to ensure that the maximum interval is less than $(\beta_n - \beta_1)/(NTYPE-1)$. NTYP is initially set to 41 and once the extra nodes have been inserted, the user is given the opportunity to change NTYP to either increase or decrease the final number of β nodes. If during the insertion process, the available storage is exceeded, the user is advised and, again, given the opportunity to change NTYP.

Once the total number of new nodes has been established and the nodes transferred to the array BETINT in INTCOM, INTPLT performs the following tasks.

- (i) ISWT is set to 1 to indicate that the data in INTCOM defines the relationship between the node indices and α, β . (Rather than the normal data used for the loss factor table.)
- (ii) The α limit arrays, ALP1IN and ALP2IN are computed according to equations (5.74) and (5.75).
- (iii) The integrand values at each mesh point are evaluated. Note that PLTINT is set to false to prevent any of the plotting facilities in FALP from being activated.
- (iv) After seeking a response from the operator, ARROUT is called, if required, to print the integrand values.
- (v) ARRPLT is called to construct the contour map.
- (vi) The β limits defined by B1 and B2 are drawn on the plot using the subroutine SMOOTH to create broken lines.
- (vii) The proof load inspection boundary is included by calling RLINE with IPM=-1.
- (viii) After seeking responses from the operator, extra R lines having $R = \alpha\psi(\beta)$ constant are constructed using RLINE.

Prompts

- (i) 'INTEGRAND PLOT?': The operator is requested to select whether to plot the current map or not. It is assumed that prompts issued prior to calling INTPLT will identify this prompt.
- (ii) 'USING NTYPE = *** INTEGRAND MATRIX IS *** BY ***
CHANGE NTYPE?':
The operator is given the opportunity to change NTYP and hence the resolution in the β direction.

- (iii) 'NEW VALUE FOR NTYPE': This prompt follows a response of 'YES' to the one above. The operator enters the new value of NTYPE.
- (iv) 'USING NTYPE = *** INTEGRAND PLOT FAILURE AT J = ***
NEW VALUE FOR NTYPE':
The available storage has been exceeded when trying to satisfy the resolution specified by NTYPE. The value of J indicates how far through the sequence of β nodes that INTPLT had got. The operator must enter a lower value for NTYPE.
- (v) 'PRINT INTEGRAND VALUES': The operator enters 'YES' is a printout of the integrand values is required.
- (vi) 'INCLUDE R LINES': The operator enters 'YES' if additional R lines are required on the plot.
- (vii) 'R VALUES': The operator is asked to specify the R values for the additional R lines. Responses follow those specified for REALIN.
- (viii) 'DOT CODE': The operator must enter an integer defining the line coding used for the additional R lines. (See SMOOTH for details.)

SUBROUTINE PLTPNT

Function

PLTPNT plots the position of the point (α, β) , (defined by ALPV and BETV in the COMMON block PARCOM), on a two-dimensional integrand diagram. The point is assumed to be associated with a double integration over α and β : a level of subdivision as indicated by the numbers J1 and J2 in INFOR1 and INFOR2 respectively can be associated with the point.

Operation

PLTPNT assumes that the plot space has been initialised by a prior call to SETPLT. The point is plotted by the subroutine POINT which is supplied by the DEC SYSTEM-10 software.

The first time PLTPNT is called, the operator is asked to specify the maximum level of subdivision for which the points will be plotted. The operator is also asked to specify the type of symbol which will be used to indicate the position of the point. The code for the symbol is stored in ICODE and is passed into POINT. The symbols and their associated code numbers are listed in Table 5.22.

Prompts

- (i) 'INDICATE MAX. LEVELS FOR PLOTTING'
- 'FUNCTION EVALUATION POINTS'
- 'BETA'

The operator is asked to specify the maximum level of subdivision in the β integration for the display of evaluation points.

- (ii) 'ALPHA': The operator is asked to specify the maximum level of subdivision for the α integration for the display of evaluation points.

Note that a value of 0 for both limits results in no action by PLTPNT. Refer to Section 4.1. for the

516

516

significance of the levels of subdivision.

- (111) 'ENTER POINT CODE - 1,2,3,4,5 OR 6': The operator is asked to specify the code for the selected symbol.

5.10.3. Loss factor maps

Contour maps of the following functions can be constructed.

(i) G Integral

The G integral is the function stored in the file GCOM.DAT and is defined by

$$GI(\alpha, \beta) = \int_{\tilde{t}_1}^{\beta} r_2(\alpha, \psi(\beta)) d\beta / \tilde{t}_r. \quad (5.263)$$

(ii) G Function

The G function is defined by,

$$GF(\alpha, \beta) = \left[r_1 \tilde{t}_1 + \int_{\tilde{t}_1}^{\beta} r_2(\alpha, \psi(\beta)) d\beta \right] / \beta \quad (5.264)$$

$$= G^*(\alpha, \beta) / \beta. \quad (5.265)$$

(iii) Loss factor

The loss factor $H(\alpha, \beta, n_0, t)$ is defined by equation (5.60) and either (5.61) or (5.62) depending on the value of n_0 .

Contour maps of the first two functions are offered by the subroutine SETTAB. The facility to construct these maps was installed during program development and it is not envisaged

that it will be used particularly often.

The loss factor has the most obvious physical interpretation and is constructed by the subroutine RSKTOT. A typical example is given in Figure 5.35.

The three functions are mapped by establishing a matrix of values at points corresponding to the intersections of the lines in the loss factor interpolation mesh. These values are calculated in the subroutine constructing the map (i.e. SETTAB or RSKTOT) and ARRPLT is called directly to construct the map. A simple switch in the subroutine P is used to distinguish between these maps and the integrand maps which use a different mesh.

An optional facility provided during the construction of the map of the G function is the construction of a representation of the loss factor interpolation mesh. Subroutine GRID makes the construction and an example was given in Figure 5.12.

UM317 LOSS FACTOR

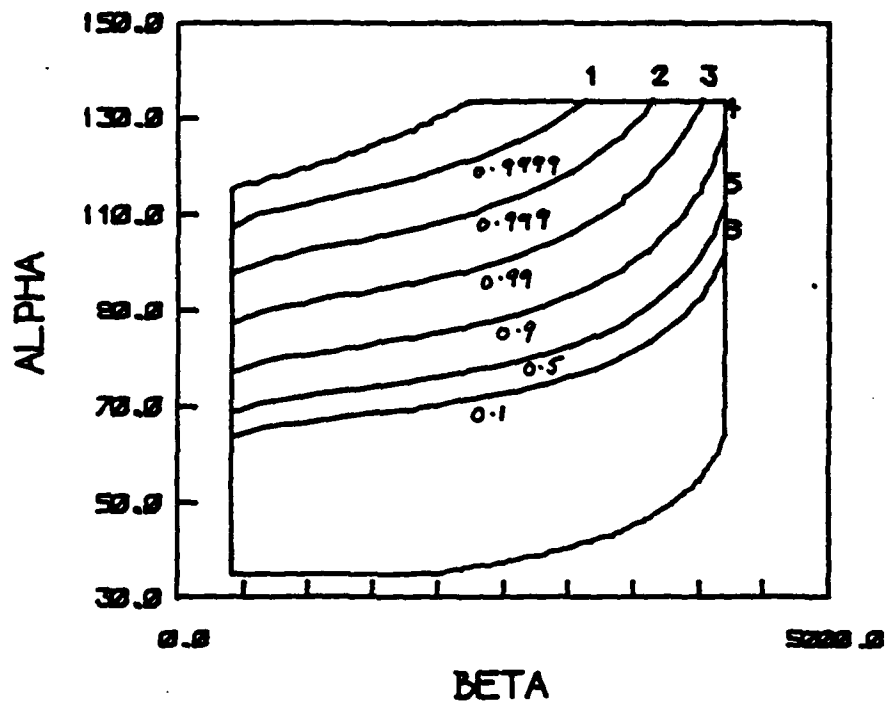


Figure 5.35. Example of a loss factor map.

SUBROUTINE ARRPLT(TITLE,NCH,ARR,M,N)

Function

ARRPLT controls the construction of a contour map of the data stored in the array ARR. This data usually represents the integrand of a two-dimensional integration over α and β , but can represent the loss factor, the G function or the G integral.

Parameter List

TITLE: Text string defining a suitable title for the contour map.

NCH: Number of characters in TITLE.

ARR: Array of data values of the function for which the map is to be constructed.

M: First dimension of ARR, (corresponding to the β direction.)

N: Second dimension of ARR, (corresponding to the α direction.)

Operation

Much of the work associated with the construction of the map is preformed by other routines which ARRPLT calls to execute the sequence of operations listed below.

- (i) A prompt identifying the map being created is generated to relate subsequent prompts correctly.
- (ii) The plot space is initialised by calling SETPLT.
- (iii) The numerical range of the data in ARRPLT is computed and used during an interactive dialogue which establishes the contour levels to be used in the map.
- (iv) The contour levels are written beneath the plot space.

- (v) If the operator selects an optional logarithmic interpolation facility, the logarithms of all the data in ARR and the contour levels are taken.
- (vi) The map is constructed by calling the subroutine CONT.

Prompts

- (i) 'FUNCTION LIMITS *****
LEVELS':
The user is requested to define the contour levels. The possible responses are described in the documentation for REALIN.
- (ii) 'USE LOG INTERPOLATION?': The user can select the optional facility for using logarithmic interpolation for the construction of the contour map.

SUBROUTINE RLINE(RARG,IPM)

Function

Subroutine RLINE draws the line $R = \alpha\psi(\beta)$ on a plot space that has been set up for a two-dimensional integrand plot. If IPM is negative, the line $R = \alpha\psi(n_0 + (\beta - n_0)t_{i_j}/t)$ is drawn instead. (The latter line corresponds to a proof load inspection boundary.)

Parameter List

RARG: Value of R for which the line is required.

IPM: Control parameter (integer).

>0 , Same as for positive values of IDOT in SMOOTH.

<0 , The alternative line described above is drawn.

IPM has the same significance as IDOT in SMOOTH.

Operation

RLINE operates by generating a sequence of data pairs of α and β which are passed to SMOOTH to construct the line. The line is vignettted by the limits (α_{\min} and α_{\max}) so that the code generating this sequence has three sections. The first and last sections determine the end points of the line. The middle section generates the intermediate values of β and the corresponding values of α .

The β points are chosen using the nodes defining $\psi(\beta)$ as a basis and inserting extra nodes to ensure that all the intervals are less than $RINTVL * (\beta_n - \beta_1)$ where RINTVL is set within RLINE. (Currently RINTVL=0.02.)

If IPM is negative, meaning that the proof load inspection boundary is being drawn, the line is restricted to $n_0 \leq \beta \leq \beta_n$.

SUBROUTINE GRID

Function

Subroutine GRID draws a representation of the mesh used for the interpolation of the loss factor on the plot space set up for the display of integrand functions.

Operation

The mesh lines correspond to integer arguments for the subroutine $P(X,Y,NPE)$. The representation is drawn in an obvious way by following lines having one of the arguments of P held at a constant value for each mesh line.

SUBROUTINE CONT(T,MD,ND,CONLEV,NC)

Function

Subroutine CONT constructs a contour map of the surface represented by the array of spot heights, T. The map is constructed relative to the two-dimensional orthogonal mesh defined by the indices of T and transformed to a real system via the subroutine P which contains all the plot system dependent code.

Parameter List

T: Array containing spot heights of the surface to be contoured.

MD: First dimension of T.

ND: Second dimension of T.

CONLEV: Array containing the contour levels to be used in the map.

NC: The number of levels stored in CONLEV.

Operation

Subroutine CONT is a large routine which was developed as part of more general purpose graphics software for which documentation will be eventually provided. It is beyond the scope of, nor required, that this documentation describes the details of the contouring process.

Broadly the array T is searched to find the significant features of the surface and possible starts and finishes of contour lines noted. For each contour level, the routine then traces the contour lines using a systematic procedure which ensures that all lines are drawn but no line drawn more than once.

Error Messages

- (1) 'ERROR ---- N (***) IS LARGER THAN IRMX (***)': The number of the second dimension of T exceeds the space available in CONT for the storage of information pertaining to the location of possible starts for contour lines. The dimension of the array IROW will have to be increased and CONT re-compiled.

Note that CONT contains another fixed array, IS, which is used to store the locations of all possible starting points for a given contour level. If this storage is exceeded, CONT automatically bands the array and constructs the map in smaller sections. If this occurs often (meaning that the surface is always very complex) the size of the array should be increased as it is slightly more efficient to construct the map for the whole array at once.

SUBROUTINE DIAG(T,M,M1,N1,I,J,C1)

Function

Diag finds the point of intersection of a contour line with the diagonal of a mesh cell. The routine is called only by CONT.

Parameter List

T: Array of spot heights.

M: First dimension of T..

M1: Local X directional offset.

N1: Local Y directional offset.

I: First index for current mesh cell.

J: Second index for the current mesh cell.

C1: Contour level.

Operation

Subroutine DIAG is called by CONT during the process of tracing a contour line. DIAG calculates the point of intersection of the contour line with the diagonal of the mesh cell and plots the point by calling subroutine P.

If the logical switch DGS is true, diagonal interpolation is neglected and DIAG does nothing.

SUBROUTINE P(X,Y,NPE)

Function

Subroutine P converts an X,Y position relative to two array indices to two real coordinates for plotting.

P is used primarily by the contouring subroutine, CONT but is also used by the subroutine GRID which constructs a representation of the interpolation mesh for the loss factor table.

Parameter List

X: X coordinate, relative to the first array index of the point to be plotted.

Y: Y coordinate, relative to the second array index of the point to be plotted.

NPE: Control parameter.

=2, Current point is joined to last point.

=3, Current point is not joined to last point. The current contour level is drawn adjacent to the point.

=4, Current point is the last point in a closed curve.

=13, As for 3 but no level is drawn.

Operation

Subroutine P has two main tasks. The first is the calculation of and from the values of X and Y. The second is the control of the plot instructions required to comply with the actions specified by NPE.

P effects the transformation from X,Y to by accessing two alternative data sources. The first defines the geometrical arrangement of the loss factor table and is stored in the COMMON blocks PSICOM and ALPCOM. The second set defines the refined geometry used to define an integrand functions and is

stored in the COMMON block INTCOM. Selection between the two data sources is made by the integer ISWT in COMMON block ISWT. If ISWT=0 the first source is used. If ISWT=1, the second is used.

In both cases, the transformation is made via the same sequence of operations. The interval containing β is found so that $\beta_j \leq \beta < \beta_{j+1}$. Then β is given by

$$\beta = \beta_j + (X - j) * (\beta_{j+1} - \beta_j). \quad (5.266)$$

The corresponding α value can then be retrieved using ALPTAB described in Section 5.4.1. Rather than obtain the α value directly, values for RK=Y are found for β_j and β_{j+1} and linear interpolation used to calculate α for β . (If direct evaluation is used, contour lines can cross the boundaries of the integration domain by, admittedly, small amounts. The result using the double evaluation and linear interpolation is aesthetically more pleasing.)

The remainder of the code performs the second task in an obvious manner.

SUBROUTINE SETPLT(TITLE,NCH)

Function

The subroutine SETPLT initialises a two-dimensional plot representing an integration domain in (α, β) space. The plot space is intended for use either as an integrand - function evaluation plot or as a loss factor map.

Parameter List

TITLE: Character string indentifying the plot.

NCH: Number of characters in the string stored in TITLE.

Operation.

The plot space is initialised via the following steps.

- (i) A new origin is established at the bottom left hand corner of the current plot space and the scale factors used by the plotting software are restored to 1.
(On an initial call to SETPLT, XVAR(1) and YVAR(1) are assumed to be set to zero so that this step has no nett effect.)
- (ii) A new plot area is established by calling SETGRF.
Note that SETPLT checks the plot counter (INP) stored in the common block PLTPMS. If INP has been incremented since the last call to SETPLT (as would be the case if PLTOUT had constructed an outer integrand plot), SETGRF is called with ISET = 1 so that the operator is asked to specify the plot size. On the first call to SETPLT the operator is advised of the and ranges and asked to specify the plot ranges for these variables and suitable numbers of interval markers for the plot space annotation (see SETGRF). The number of intervals is selected by specifying a suitable 'step size' for each variable.

- (iii) Scale factors are computed and the origin reset so that the bottom left hand corner of the plot space corresponds to the point defined by the minimum values of α and β selected by the operator. The scale factor used by the plotting software is set so that future plotting can be done directly in terms of α and β . The subroutine SCLFCT is called to set scaling parameters used by SMOOTH.
- (iv) A representation of the maximum possible limits for the integration domain for cracked structures is constructed using the lines $\alpha = \alpha_{\min}$, $\alpha = \alpha_{\max}$, $\beta = \bar{\epsilon}_1$, $\beta = \bar{\epsilon}_f$, $\alpha = R_{\min}/\psi(\beta)$ and $\alpha = R_{\max}/\psi(\beta)$.

Prompts

- (i) 'INITIALISE INTEGRAND PLOTS'
 'BETA VALUES BETWEEN **** AND *****'
 'BETAMIN,BETAMAX,BETASTEP': Request for the range of β to be used to set up the plot space and the interval to be used between 'tic marks' on the β axis.
- (ii) 'ALPHA VALUES BETWEEN **** AND *****'
 'ALPMIN,ALPMAX,ALPSTEP': Request for the range of α to be used to set up the plot space and the interval to be used between 'tic marks' on the α axis.

Common variables changed

INP: Graph counter.

XVAR: Array of parameters for horizontal axis.

YVAR: Array of parameters for vertical axis.

5.10.4. Graphics support subroutines

The subroutines described in this Section provide basic graphics support as described below.

(i) Initialisation of a plot area

Each graphical construction requires a 'plot space' to be initialised. The subroutine SETGRF performs this task and is used by the graphical routines in NERF and NERPLT. As currently configured, the graphs are drawn to fit on A4 size 'pages'.

(ii) Smoothing and line coding

The subroutine SMOOTH allows a certain amount of smoothing to be applied to a curve defined by a sequence of line segments. A by-product of this process is the ability to draw the smoothed line as a broken line with variable mark-space ratios. This facility is used by NERF to draw smooth representations of the constant R lines on the integrand maps and to provide line coding which is used by NERPLT to distinguish between functions.

Subroutine SMOOTH is supported by the subroutines PLOTD and SCLFCT.

(iii) Conversion of numbers to character strings

The annotation of a graph requires the conversion of

532

532

numbers to character strings. This facility is provided
by the subroutine FSIZE.

SUBROUTINE FSIZE(ARG,TXT,LENGTH)

Function

FSIZE converts a number stored in ARG to a text string suitable for display on a plot. The main task executed by FSIZE is the selection of an appropriate format for the number.

Operation

ARG: A real number for conversion to a text string.

TXT: Array into which the text string will be returned to the calling code.

LENGTH: The number of characters returned in TXT.

Operation

The format is selected according to the following rules.

- (1) ARG = 0

The format selected is F4.1.

- (11) $10^{-5} < \text{ARG} < 10^5$

Fixed decimal format is used. The format is F(L).(INDD) where

$$L = \text{INDF} + \text{INDD} + 2.$$

(5.247)

INDF is the number of digits to the left of the decimal point. INDD is the number of digits to the right. INDD is chosen such that a maximum of NSIG significant digits are used to represent the number with all trailing zeros suppressed. NSIG is set as a parameter in FSIZE. (Currently NSIG = 8.)

If ARG is negative the length L is increased by 1 to include the sign.

534

534

(111) ARG >10⁵ or ARG <10⁻⁵

Exponential format is used, being E11.4 if the number is negative and E12.4 if the number is positive.

Note that all the formats allow for a single space at the front of the text string.

SUBROUTINE PLOTD(X,Y,NPE)

Function

The sole function of PLOTD is to provide an interface between the plotting requirements of subroutine SMOOTH and the subroutine PLOT provided on the A.R.L. DEC system 10.

Parameter List

X: X coordinate of the point to be plotted.

Y: Y coordinate of the point to be plotted.

NPE: Pen control parameter.

-3, Point is not joined to last point.

-2, Point is joined to last point.

Operation

PLOTD changes the pen control parameter from 2 to 4 and provides the plot logical unit number (5) for transfer to the subroutine PLOT.

SUBROUTINE SETGRF(LU,XVAR,YVAR,LX,LY,TITLE,NX,NY,NT,ISSET)

Function

The subroutine SETGRF sets up a plotting area on the file associated with the logical unit, LU. The plotting area is a sub-area of a page. SETGRF draws an outline of the plotting area, annotates the axes and exits with the origin set at the bottom left hand corner of the plotting area and plotting coordinates scaled so that one unit corresponds to one inch on the plot.

Parameter List

LU: Logical unit number.

XVAR: Array containing three variables:

XAR(1) Minimum value for the coordinate represented by the horizontal axis.

XAR(2) Maximum value for the coordinate represented by the vertical axis.

XAR(3) Size of interval between 'tic' marks on the horizontal axis.

YVAR: Array containing three variables as described above for XVAR, but pertaining to the vertical axis.

LX: Character string defining a label for the horizontal axis.

LY: Character string defining a label for the vertical axis.

TITLE: Character string defining a title for the plot.

NX: Number of characters in the string in LX.

NY: Number of characters in the string in LY.

NT: Number of characters in the string in TITLE.

ISSET: If ISSET is non zero, SETGRF will ask the operator to specify the plot size.

If ISSET is zero, the operator will not be asked and SETGRF will use the parameters established during a previous call.

Operation

SETGRF establishes a plot area according to the variables defined in Figure 5.36. The size of the page (XLR x YLR) and the minimum margins (XMARH, XMARL, YMARH and YMARL) are defined with the subroutine and are currently set to correspond to an A4 page. YC is set to correspond to the outer extremities of marks that are drawn on the plot to delimit each A4 page: the size of these marks is set by BIT. The title is positioned above the plot area by a distance determined by YTITLE.

Other parameters set within SETGRF are: CHRBIT, half the size of the smallest character drawn by the plotting software supplied by the system; MAG, the magnification of that character size to generate the numerical labels for the coordinate axes, and SMALL, the length of the 'tic marks' placed on each axis.

For reference, a typical plot initialised by SETGRF is shown in Figure 5.37.

The initialisation process follows the following sequence.

- (i) If ISET is non-zero, the operator is asked to specify the size of the plot in inches. This action specifies values for XL and YL. From these values the variables XD and YD are calculated so that the plot space is positioned centrally in the vertical direction and right justified against the margin in the horizontal direction.
- (ii) On entry, the plotter pen is moved to the midpoint of the left hand edge of the page. The A4 separating marks are then drawn.
- (iii) The outline of the plot space is constructed and the origin shifted to the bottom left hand corner.
- (iv) The title is placed centrally above the plot space.
- (v) The limits of the horizontal axis are annotated with representations of the values specified by XVAR.

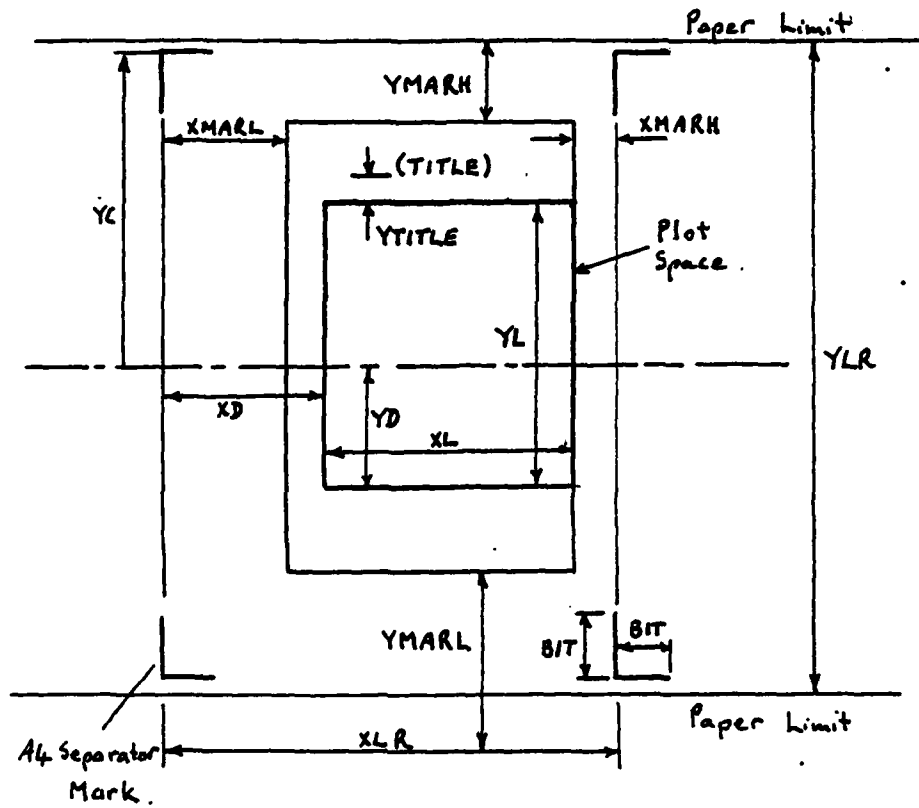


Figure 5.36. Program variable names and their associated measurements as used by SETGRF to set up a plot space.

UM324 RSLT TERM

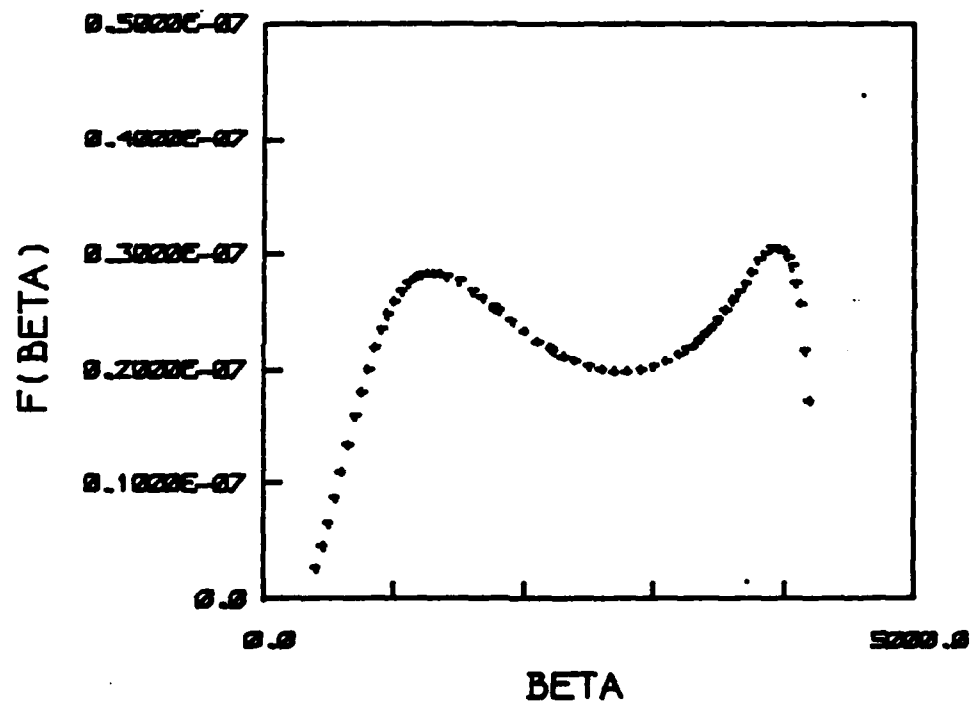


Figure 5.37. Example of a plot initialised by SETGRF.
(Outer integrand plot for $r_s(t)$ for
example A, $t = 1200$.)

- (vi) Vertical 'tic marks' are inserted along the horizontal axis. The number of marks is set by the size of the interval specified by XVAR(3). If XVAR(3) = 0, 10 intervals are used by default. The number of intervals is limited to 20.
- (vii) The label for the horizontal axis is placed, centrally below the plot space.
- (viii) Operations (v) to (vii) are repeated for the vertical axis. This time, each 'tic mark' is annotated with a representation of the numerical value of the coordinate at that location on the vertical axis.

Note that the character strings representing values on the coordinate axes are produced by the subroutine FSIZE.

The example given in Figure 5.3 was produced by a call of the form,

```
CALL SETGRF(3,XVAR,YVAR,'BETA','F(BETA)','UM324 RSLT TERM',4,7,15,1)
```

with XVAR(1) = 0, XVAR(2) = 5000, XVAR(3) = 1000, YVAR(1) = 0, YVAR(2) = 5e-8 and YVAR(3) = 1e-8. The operator chose, XL = 4, YL = 3.5.

Prompts

- (i) 'LENGTH OF X AXIS (INCHES)': Request for XL.
- (ii) 'LENGTH OF Y AXIS (INCHES)': Request for YL.

Common Variables Changed

XO: Minimum value for x for coordinate system for plot space (set to 0).
 XL: Maximum value for x for plot space coordinates.
 YO: As for XO, but for y.
 YL: As for XL, but for y.

SUBROUTINE SCLFCT(LU,SX,SY)

Function

The subroutine SMOOTH requires the current scale factors that are in use by the plotting system to be stored in the COMMON block SMTHSC. Subroutine SCLFCT estimates those scale factors and transfers them to the COMMON block. The scale factors are also returned to the calling code via the parameter list.

Parameter List

LU: Logical unit number for the plot file.

SX: Scale factor for X as transferred to COMMON.

SY: Scale factor for Y as transferred to COMMON.

Operation

The scale factors are estimated by moving the plotter a fixed distance and accessing the plot software to find the real coordinates of the end points of movement. On the DEC system 10, the real coordinates are returned by the subroutine WHERE and the movement is effected by calling PLOT with integer arguments of 100 for X and Y. This results in a pen movement of 1 inch in each direction. A call with X and Y equal to -100 restores the plotter to the position prior to the call to SMOOTH.

SUBROUTINE SMOOTH(XP,YP,NPE,IDOT)

Function

Subroutine SMOOTH provides a curve smoothing facility for a line defined by a sequence of data pairs (XP,YP) to which SMOOTH has access only at the time of call.

A secondary function (but the primary use made by NERF) is to provide line coding facilities by inserting breaks in the line according to the code in IDOT.

Parameter List

XP: X coordinate of the point on the line.

YP: Y coordinate of the point on the line.

NPE: PLOT control parameter.

=2, Point is joined to the last point received by SMOOTH.

=3, Point is not joined to the last point received by SMOOTH. (i.e. point is the first point of a new line.)

=4, Point is the last point of closed curve.

=5, Current line is complete; XP and YP do not define a new point.

=6, Smooth asks the operator to specify a new smoothing (or line break) interval.

IDOT: Line coding control parameter.

=0, Line is solid. No breaks are inserted.

>0, Line is broken. Breaks equal to the current smoothing interval are inserted between solid sections that are IDOT*(the smoothing interval) long.

<0, As above but each second solid section is only one smoothing interval long. (i.e. dash-dot lines are drawn.

<-19, No smoothing is selected. The line is constructed in the same way as if the normal plot routines had been called directly.

Operation

The smoothing process used by SMOOTH is based on fitting cubic polynomials through the data points in such a way that the resulting line has tangential continuity at each point. Because SMOOTH does not retain all the data pairs defining a continuous line, the resulting curve is close to but not equivalent to a cubic spline.

The points created during the smoothing process are spaced at approximately equal intervals along the line and hence line coding is a facility which can be included for a trivial cost in terms of coding complexity.

The smoothing interval and hence the break interval is specified in terms of plot inches. This interval is set during the first call to SMOOTH by the same interactive dialogue which changes the interval in response to a call with NPE=6. Because the plot routines may have implicit scale factors operative, scale factors for X and Y are assumed to be available in SCX and SCY in the COMMON block SMTHSC. If these factors are 1 or 0, the scale factor logic is bypassed. (Subroutine SCLFCT can provide the current scale factors if required.)

Note that all calls to the plot software are via the subroutine PLOTD so that SMOOTH is independent of the plot system on which NERF is installed.

544

544

Prompts

- (1) 'SMOOTHING INTERVAL': Prompt issued by the section of code which provides the facility for changing the smoothing interval.

5.10.5. The graphics post processor NERPLT

The program NERPLT was designed primarily for constructing graphs of the reliability functions calculated by NERF. These functions are written by NERF into a file that has essentially the same format as the function files used to define the input functions. It was thus a trivial matter to modify NERPLT so that it could plot a graph defined by any function file so that, in principle, the input functions could also be plotted. Subsequently, this facility was extended by allowing NERPLT to access the various functions in NERF to construct the input functions (including those that do not rely on function files) directly.

NERPLT now provides facilities for constructing graphs of the NERF input functions, including the density functions, together with any functions represented by function files. The latter functions can be mixed on a single graph to produce a 'multi-function' plot. A multi-function plot uses the first function as a basis for setting up the graph and establishing scale factors. Subsequent functions are plotted using the established plot space and the same scale factors. In this way, the results from several runs of the program NERF can be presented on the same graph.

Examples of the output produced by NERPLT were given in Chapter 3 when describing the input functions and the various inspection procedures modelled by NERF.

It is not intended that the documentation presented here is complete, especially in terms of the operation and prompt messages issued by NERPLT. These details are covered more completely by Mallinson and Graham¹³. The documentation presented here covers the operation of the various subroutines which are not described in the user manual.

The operational aspects of NERPLT are determined by the main program, PROGRAM NERPLT which executes the tasks portrayed schematically in Figure 5.38. The subroutine READMF reads the function files and the subroutine FPLOTS uses the NERF functions to set up sequences of ordered pairs that represent the input functions.

Note that the subroutine FPLOTS, in effect, constructs a sequence of ordered pairs which represents the interpolated function derived from the function file data for input functions so defined. The graphical output is thus a way of checking the performance of the interpolation procedures. The subroutine FPOINT constructs graphical representations of the points actually used to define one of these functions. (See Figure 3.1, for example.)

Plotting a sequence of ordered pairs is controlled by the subroutine PLTS. Because NERPLT allows the user to define the scale factors and plot limits, the data must be 'windowed' so that the portion of the function that fits within the plot space is the only section of the function actually plotted. This

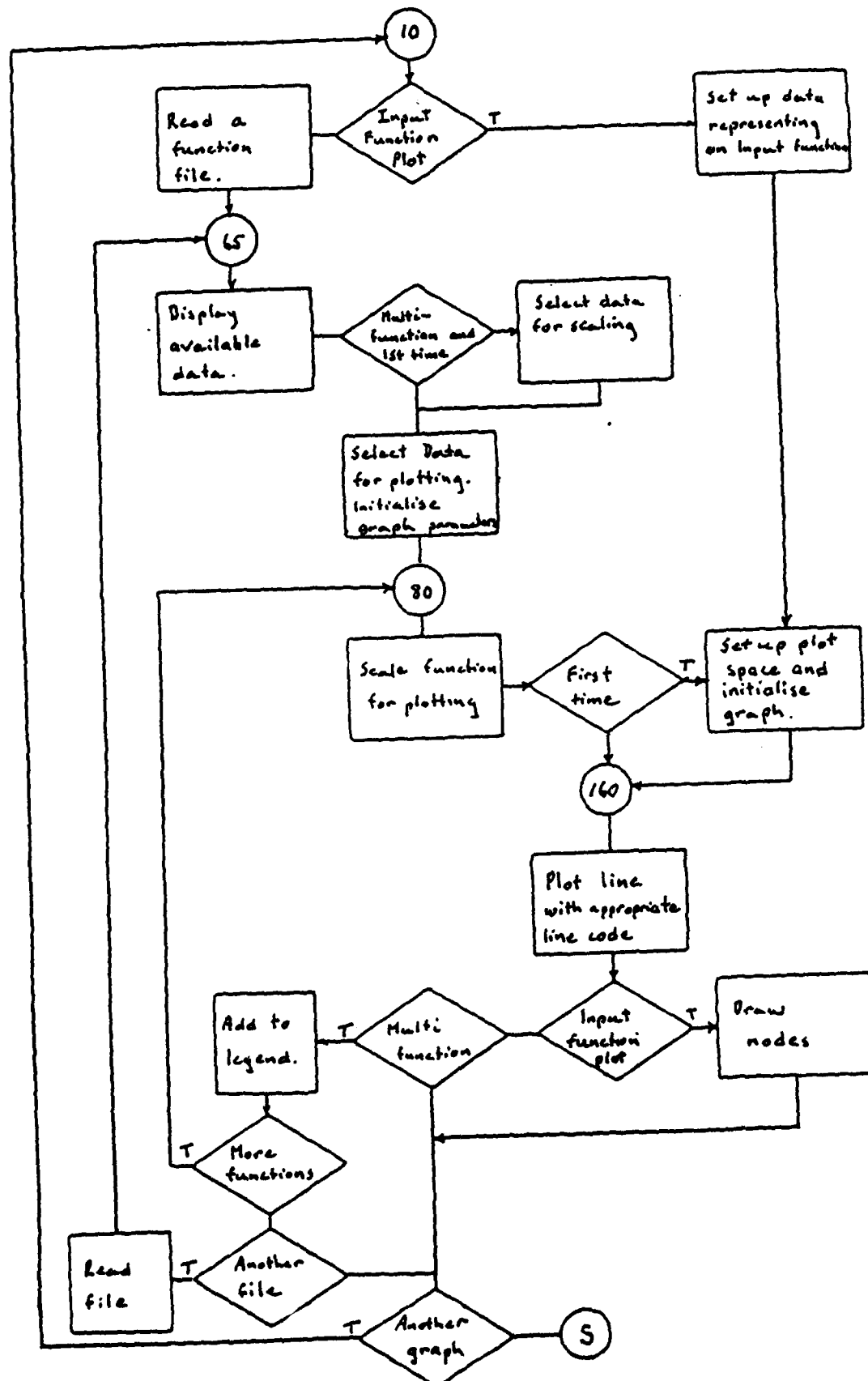


Figure 5.38. Logic (simplified) in PROGRAM MERPLT

windowing facility is provided by the subroutines WINDOW, and COMPRS.

The rest of the code used by NERPLT is supplied by the NERF library of subroutines. Note that the current implementation of NERPLT contains special forms of the subroutines PROMPT and ECHO. These routines do not write any information to the secondary output file as is the case for the routines in the main NERF library. If a record of the dialogue is required, these subroutines could be modified accordingly.

PROGRAM NERPLT

Function

Program NERPLT provides an interactive facility for the construction of graphs of the NERF input functions and the reliability functions computed by NERF.

Operation

The sequence of operations controlled by NERPLT is described schematically in Figure 5.38. The code broadly follows the logic shown in that Figure which together with the operational description given by Mallinson and Graham¹³ should be sufficient for a FORTRAN programmer to understand the code. The following points are worthy of mention.

- (i) The logical variables listed below are used to control the sequence of operations.

MPLT: If true, the multi-function plot facility has been selected.

ADDPLT: If true, the next function will be added to the existing graph.

FPLTS: The current graph is a NERF input function. Multi-function plots are not permitted if this is the case.

APLT If true, at least one graph has been initialised so that basic input/output operations have been performed.

TIME: If true, and the graph is multi-function, the current function is the first on the graph.

(ii) The NERF input functions are initialised by calling CFIN which reads the control file and initialises the density functions and the various functions that are defined by function files. In this way, the input files pertain to the run defined by the current control file.

(iii) The output file, containing the graphical information, has the name 'NAME.DPT' where the string denoted by NAME depends on the first function actually plotted. If the first function is one of the input functions, NAME corresponds to the run identification in the control file; otherwise it corresponds to the name of the first file read by READMF.

(iv) The subroutine READMF is similar to READFN described in Section 5.9.3, with the exceptions that the operator is asked to specify the file name for the function file and that several columns of data can be read from the file and are interpreted as defining several functions for the same set of argument values.

SUBROUTINE COMPRS(X,Y,X1,Y1)

Function

Given a point (X,Y) inside the plot space and a point (X1,Y1) outside the plots space, COMPRS finds the coordinates of the intersection of the line joining (X,Y) and (X1,Y1) with a boundary of the plot space.

Parameter List

X: X coordinate of the point inside the plot space.

Y: Y coordinate of the point inside the plot space.

X1: X coordinate of the point outside the plot space.

Returned as the X coordinate of the intersection point.

Y1: Y coordinate of the point outside the plot space.

Returned as the Y coordinate of the intersection point.

Operation

The line joining (X,Y) and (X1,Y1) is compressed, first in the X direction, by computing the intersection with $x = X_0$ or $x = X_L$. It is then compressed in the y direction by computing the intersection with $y = Y_0$ or $y = Y_L$.

SUBROUTINE FPLOTS(LU,IFUNC,XMIN,XMAX,YMIN,YMAX,XLAB,YLAB,X,FY)

Function

The subroutine FPLOTS sets up a sequence of ordered pairs of argument and function values of one of the NERF input functions. This sequence is intended for subsequent plotting by NERPLT, and is sufficient to define a smooth curve representing the function. The generated sequence is not the same as that which may be used to define an interpolated function.

Parameter List

LU: Logical unit number for the plot file, (not used).

IFUNC: Integer defining the input function . (See parameter list for FUNC.) Returned to calling code.

XMIN: Lower limit for the range of the argument, returned to calling code.

XMAX: Upper limit for the range of the argument, returned to calling code.

YMIN: Lower limit for the range of the function, returned to calling code.

YMAX: Upper limit for the range of the function, returned to calling code.

XLAB: Label for the horizontal axis of the graph, returned to calling code.

YLAB: Label for the vertical axis of the graph, returned to calling code.

X: Array containing the generated argument values, returned to calling code.

FY: Array containing the generated function values, returned to calling code.

Operation

The operator is asked to select an input function. A valid function must be selected and the program will loop until a correct entry is made by the operator.

Once an input function has been selected, its range is determined and values for XMIN, XMAX, YMIN, YMAX computed. The sequence of ordered pairs is then generated in a straightforward manner. Note that if a logarithmic plot has been selected that logarithms of the function are taken before the sequence is returned to the calling code.

Currently 200 ordered pairs are used to define the function.

If a density function is selected and the associated random variable is being neglected from the model, the sequence is not generated and IFUNC is returned with the value, 99.

SUBROUTINE FPOINT(LU,XVAR,YVAR,IFUNC)

Function

FPOINT constructs representations of the locations of the nodes used to define the NERF input functions. In the current implementation the nodes are represented by small squares centred at the points defined by the sequence of ordered pairs of argument and function values upon which each input function is based.

Parameter List

LU: Logical unit number for the file containing the graphical output.

XVAR: Array containing the minimum value, maximum value and increment for axis marking for the argument.

YVAR: Array containing the minimum value, maximum value and increment for axis marking for the function.

IFUNC: Integer identifying the input function.

1. Risk rate function, $r_2(R) = \bar{P}_L(R) \cdot l_r$.
2. Relative strength function, $\psi(\bar{t})$.
5. Crack growth function, $a(\bar{t})$.
6. Inverse crack growth function, $a^{-1}(a)$.
8. Strength decay function, $R(a)$.

Operation

The assumption is made that the graph has been initialised and the contents of XVAR and YVAR are correct for the function being plotted. Each function is treated separately. The appropriate array of argument values (e.g. BETA for \bar{t} values for $\psi(\bar{t})$) is accessed. For each argument value, the corresponding function value is evaluated and the point constructed if within the confines of the graph, as determined by the parameters XO, XL, YO and YL in the common block AREAXY.

FUNCTION FUNC(X,IFUNC)

Function

The function FUNC returns a value of an input function for a given value of the argument, X. The particular input function is selected by the value of the parameter IFUNC. This function helps make higher levels of the graph plotting software less dependent on the precise input function selected.

Parameter List

X: Value of the argument for which the function evaluation is required.

IFUNC: Integer defining the input function.

1. Risk rate function, $r_2(R)$.
2. Relative strength function $\psi(\tilde{t})$.
3. Density for virgin strength $p_{\mu}(\alpha)$.
4. Density for age, $p_{\beta}(\beta)$.
5. Crack growth function, $a(\tilde{t})$.
6. Inverse crack growth function, $a^{-1}(a)$.
7. Density for initial crack length $p_{a_0}(a_0)$.
8. Strength decay function, $R(a)$.

Operation

FUNC simply calls the appropriate function sub-program to evaluate the function selected by IFUNC.

SUBROUTINE PLTS(XA,F,NUM,DOTS)

Function

The subroutine PLTS controls the plotting of the curve defined by the sequence of ordered pairs stored in XA and F.

Parameter List

XA: Array of values of the argument, or variable corresponding to the horizontal axis of the graph.

F: Array of values of the function, or variable corresponding to the vertical axis of the graph.

NUM: Number of ordered pairs defining the curve.

DOTS: Parameter controlling the line code used for the curve.

=0, the line is solid.

>0, Line is broken with every second solid section being DOTS times as long as the intervening breaks.

<0, Line is broken. Solid sections are DOTS times as long as the intervening breaks.

This convention is similar to that used for subroutine SMOOTH with the exception that the sign of the control parameter is reversed.

Operation

The curve is constructed by a sequence of calls to WINDOW followed by a final call to SMOOTH with NPE=5, to complete the curve (see documentation for SMOOTH.)

If two adjacent values of XA are identical, then pen up instructions are issued to WINDOW to ensure that a possible discontinuity in the function is properly represented.

SUBROUTINE WINDOW(X,Y,NPE,IDOT)

Function

The subroutine WINDOW provides a windowing facility whereby a sequence of plotted points can be vignettted by imposed limits for plotting. If the sequence of points represents a continuous line sequence, WINDOW ensures that the segment between a point within the 'window' and one outside is correctly drawn so that it appears to be 'cut' by the plot boundary.

The subroutine is called instead of the usual plotting routines, or the subroutine SMOOTH. These routines are called by WINDOW after the clipping process has been completed.

Parameter List

X: X coordinate of the next point to be plotted.

Y: Y coordinate of the next point to be plotted.

NPE: Pen control parameter.

=2, the point defined by (X,Y) is joined to the last point.

≠2, the point defined by (X,Y) is not joined to the last point.

All other values of NPE are ignored and are passed directly to SMOOTH.

IDOT: Line coding parameter. Passed directly to SMOOTH: refer to the documentation for that subroutine for details.

Operation

The coordinate limits for the plot space are assumed to be initialised and stored in the common block AREAXY. The limits for x are defined by XO and XL and those for y by YO and YL.

The point (X,Y) is tested against the coordinate limits and the following conditions are identified.

- (i) (X,Y) inside window; last point inside. No window operation is required. (X,Y) is stored as (X2,Y2) for later use.
- (ii) (X,Y) inside window; last point outside. If points are joined, use COMPMS to find the intersection of the line joining (X1,Y1) and (X,Y) with a boundary of the plot space. Plot the intersection with a pen up instruction and draw the line joining it to the point (X,Y). Store (X,Y) as (X2,Y2).
- (iii) (X,Y) outside window; last point inside. If points are joined, call COMPMS to find the point of intersection of the line between (X2,Y2) and (X,Y) with a boundary of the plot space. Draw the line between (X2,Y2) and that point. Store (X,Y) as (X1,Y1).
- (iv) (X,Y) outside window; last point outside. Store (X,Y) as (X1,Y1).

Local variables in WINDOW have the following meanings.

- X1,Y1 Coordinates of the last point outside the window.
- X2,Y2 Coordinates of the last point inside the window.
- X3,Y3 Coordinates of the intersection of the line leaving the window and a boundary of the plot space.
- ICOMP Flag indicating whether the current point is the first of a sequence of points inside the window.
(ICOMP=0, point is the first outside; ICOMP = 1, point is not the first outside.)

5.11. The Data Preparation Program, NERPRE

The data preparation program NERPRE is, in fact, additional to the essential requirements for the operation of NERF. The control file and function files can be prepared using the text editing facilities provided by the operating system on which NERF is installed. NERPRE provides an alternative method for the preparation of input data which has the advantage that the user does not have to learn how to operate the text editor or the format requirements of the control and function files.

As was the case with NERPLT, much of the operational details of NERPRE are described in the user manual (Mallinson and Graham¹³). The documentation presented here concentrates on details of the subroutines not covered by the user manual. The documentation is not intended to be complete, but should be sufficient for a FORTRAN programmer to understand the code.

Much of the operational character of NERPRE is determined by the code in the main program, PROGRAM NERPRE, which is structured to provide multiple options which are selected, one at a time, by the user. The essential operations are shown schematically in Figure 5.39. The subroutine SELECT performs the tasks of asking the user to specify the next action to be taken, identifying the action and returning a flag with a value that can be used by PROGRAM NERPRE to jump to the appropriate section of code. When the action is finished, control returns to the point where the subroutine SELECT is called to choose a new action.

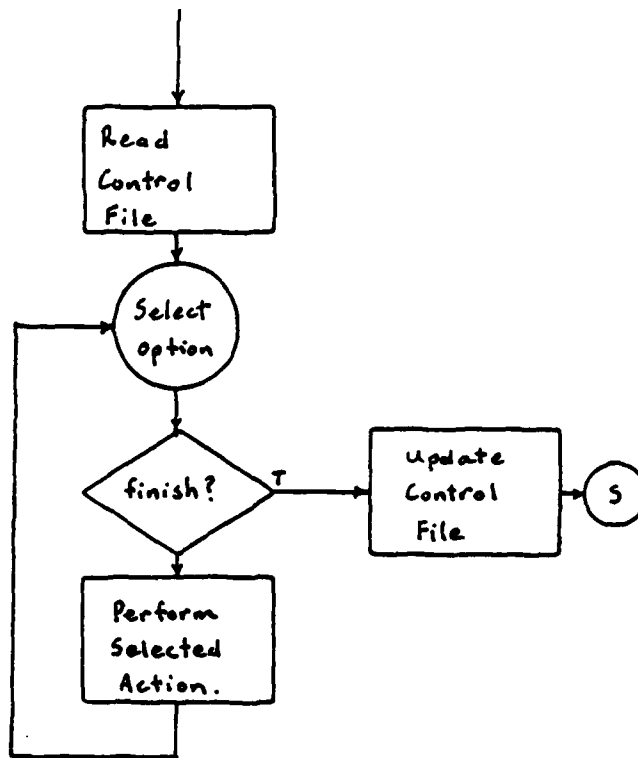


Figure 5.39. Simplified flow of logic in NERPRE.

When the program first starts execution, a check that a control file already exists is made. If a file does not exist, NERPRE executes every data preparation action to create an appropriate set of data before calling SELECT. This alternative mode of operation is controlled by the logical variable EDIT which if false suppresses the return to the call to SELECT after each action. Instead control passes to the next section of code. This continues until the last action has been made and control unconditionally returns to the subroutine SELECT. This 'non-editing' mode can also be invoked by the user selecting the option 'NEW FILE' which results in EDIT being set false as if a control file did not exist.

An exception to the simple process illustrated in Figure 5.39 occurs if the option 'FUNCTION FILE' is selected. The operation in NERPRE follows the simple sequence of performing an action and returning to call SELECT. However, the subroutine FCNFIL, which is called in the process, contains its own set of options and executes a selection process which to the user appears as a sub-set of the overall option selecting process. While in FCNFIL, none of the general options are available and while in PROGRAM NERPRE, none of the options in FCNFIL are accessible.

Another exception is, of course, the code associated with the option 'FINISH'. The subroutine CFOUT is called to write a new control file and the execution of NERPRE ceases.

PROGRAM NERPRE and the subroutine SELECT provide most of the option selecting and control logic. These two code modules are

described in this Section. The subroutines associated with the preparation and editing of a control file are described in Section 5.11.1 and those associated with the preparation of function files are described in Section 5.11.2.

PROGRAM NERPRE

Function

PROGRAM NERPRE provides an interactive facility for the preparation and editing of control and function files for input to NERF.

Operation

As described in the text of Section 5.11, the operation of NERPRE follows the simple option selecting and action performing logic shown in Figure 5.39. The heart of the code, as far as this aspect is concerned is the computed GO TO statement following the call to SELECT. The text identifying each option is, of course, stored within SELECT. However, the same text strings have been placed in comments heading each section of code to make identification of the code associated with each option easy.

The object of the editing or preparation process is to ensure that the variables in the various common blocks have the correct values. These variables have the same names and meanings as those used in NERF. Input from and output to the control file are controlled by the subroutines CFIN and CFOUT which are supplied by the NERF library. The code in NERPRE does not, therefore, directly influence the format of the control file.

The code associated with each option performs the allotted tasks in an obvious manner.

SUBROUTINE SELECT(TXT,NCH,IOPT)

Function

The subroutine SELECT controls the selection of one of the options provided by NERPRE by seeking a response from the operator and setting IOPT accordingly.

Parameter List

TXT: Character string containing a suitable request for an operator response.

NCH: Number of characters in the string stored in TXT.

IOPT: Integer which is set to a value that will identify the option selected by the operator.

Operation

The possible user responses are stored in the array RESPON. Up to 15 characters can be used to identify an option. The array NCHR contains the number of characters actually used to identify each response.

In the current version, the response is tested word by word rather than character by character so that at least 5 characters must be correct for a response to be identified.

The response 'LIST' will result in all the correct responses being listed on the remote terminal.

5.11.1. Preparation and editing of the control file

Much of the code in PROGRAM NERPRE is associated with the manipulation of the data in the common blocks that will eventually be written into the control file by the subroutine CROUT. A given option may change a single variable or a whole group of variables, depending on the nature of the data stored in those variables.

In the majority of cases, the subroutine CHANGE is called to manipulate the data. This subroutine detects whether the logical variable EDIT is true and if so can inform the operator of the contents of the variables before new values are entered. CHANGE is actually an entry point in the subroutine SELECT. This means that the text stored in the array RESPON is available for the construction of the prompt messages.

The parameters for the density functions are manipulated by the subroutine PDFCNG. This localises the code associated with the density functions so that extensions to the library of standard functions evaluated by NERF can be easily extended.

Some variables, such as the run identification and the logical switches are manipulated directly by code in NERPRE. It is assumed that the operator will have no interest in the existing values of these variables and no facility for informing the operator of their values is provided.

NERF A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION
OF RELIABILITY FUN. (U) AERONAUTICAL RESEARCH LABS
MELBOURNE (AUSTRALIA) G D MALLINSON ET AL. SEP 83
ARI/STRUC 397 1/6, 9/2

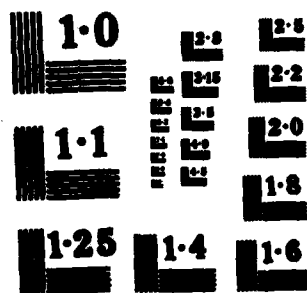
7/7

1946 1 A.S., 11 140

1/6, 4/2

44

END
DATE
FILMED
10 84



The option 'CHECK' allows the operator to scan all the data that will be transferred to the control file. The data is displayed in a meaningful way on the remote terminal by the subroutine CHECK.

Note that NERPRE makes liberal use of the subroutines TXTIN, INTGIN, REALIN and PROMPT together with the function INTEST. These, with the exception of PROMPT are obtained from the main NERF library.

As was the case with NERPLT, NERPRE has its own versions of PROMPT and ECHO to suppress the output of information to the secondary file.

SUBROUTINE CHANGE(TXT2,NCH2,VAR,NVAR,NTYPE)

Function

The subroutine CHANGE controls the manipulation of the data in the array VAR.

Parameter List

TXT2: Text string identifying the editing operation.

NCH2: Number of characters in the string stored in TXT2.
If NCH2 is zero, the identification prompt may be constructed from the text strings stored in RESPON.
(CHANGE is an entry point in SELECT.)

VAR: Array containing the variables to be changed.

NVAR: Number of variables in VAR.
If NVAR is zero, then CHANGE will select the number of possible variables from information stored within the routine and accept as many values as the operator specifies.

NTYPE: Integer identifying the editing action. (Generally, it corresponds to the value of IOPT returned by SELECT.)

Operation

The operation of the subroutine depends on the status of the logical variables EDIT and VALUE.

- (1) If EDIT is false, the user is given no choice and the data in VAR is overwritten. If NCH2 is zero, the appropriate text string in RESPON is output to identify the data required.

For those cases where the required data are a sequence of numerical values of unspecified number, data stored

in NVLIM is accessed to determine the maximum permissible number. The subroutine REALIN is then used to interactively construct the sequence.

- (11) If EDIT is true, VALUE is accessed and if required, the existing numerical data in VAL are displayed and the operator given the chance to negate the editing action. Otherwise, the data is renewed using the same methods as apply when EDIT is false.

SUBROUTINE CHECK

Function

The subroutine CHECK displays the information that will be written to the control file in an informative manner on the remote terminal.

Operation

The bulk of the code in CHECK is concerned with the construction of prompts which will inform the operator of the values of the variables in the common blocks that are accessed by CROUT. These prompts are designed to be informative in a way that allows a ready check of the accuracy of the data.

The amount of data is too much for it all to be displayed on a VDU screen at the same time. CHECK consequently displays a section of the data and then pauses, waiting for a user response before continuing. This pause is provided by the subroutine PAUSCT. It is envisaged that the code associated with effecting this pause could be machine dependent and a separate subroutine was provided. The subroutine is very short and no separate documentation is required.

SUBROUTINE PDFCNG(TXTLAB,NCH,K,C1)

Function

The subroutine PDFCNG provides an editing facility for the parameters defining the density functions.

Parameter List

TXTLAB: Text string identifying the random variable for which the parameters pertain.

NCH: Number of characters in the string in TXTLAB.

K: Integer identifying the type of standard density function.

1. Log normal.
2. Extreme value.
3. Gamma.

C1: 3 word array.

C1(1): Dispersion parameter (k).

C1(2): Minimum value parameter (e).

C1(3): Location parameter (v).

Operation

The logical variables EDIT and VALUE are accessed. If both are true, PDFOUT is called to display the parameters. The operator is asked if the existing values are acceptable. If they are, no further action is taken; otherwise the operator is asked to specify, first the type of density function and then the values of the appropriate parameters.

571

571

SUBROUTINE PDFOUT(K,C1)

Function

The subroutine PDFOUT displays the parameters defining a density function on the remote terminal.

Parameter List

K: Integer defining the standard distribution.
1. Log normal.
2. Extreme value.
3. Gamma.

C1: 3 word array.
C1(1): Dispersion parameter (k).
C1(2): Minimum value parameter (e).
C1(3): Location parameter (v).

Operation

PDFOUT identifies the type of distribution and then constructs an appropriate informative message.

SUBROUTINE VALOUT(ARR,N)

Function

The subroutine VALOUT displays the numerical data stored in ARR on the remote terminal.

Parameter List

ARR: Array containing data to be displayed.

N: Number of variables stored in ARR.

Operation

The code in VALOUT constructs and displays prompts that are appropriate to the amount of data stored in ARR.

5.11.2. Preparation of function files

The preparation of function files is controlled, almost entirely by the subroutine FCNFIL which is called by NERPRE in response to the selection of the option 'FUNCTION FILE'.

FCNFIL operates a sub-option selection process which allows function files to be created or edited. The format of the files corresponds to that specified by the subroutine READFN described in Section 5.9.3.

On entry to FCNFIL, the operator is asked to select whether an existing file is to be edited or a new one created. In the first case, the file is accessed and its contents read into core. In the second case, a dialogue prompts the operator to enter the initial set of data for the new file. After receiving the header text, the subroutine DATIN is called to input the sequence of ordered pairs.

After the function file data has been initialised by the above actions, the operator is advised of the editing options available. These are described in the description of the subroutine FCNFIL below.

Note that, as the code is currently constructed, the editing operations can only operate on the data initialised as FCNFIL is entered. To change the data by, for example, reading a new file, the option 'SAVE' must be selected and control returned to the main program. FCNFIL can then be re-entered and the data changed.

SUBROUTINE DATIN(X,F,NUM)

Function

The subroutine DATIN controls the entry of a sequence of ordered pairs from the keyboard of the remote terminal or VDU.

Parameter List

X: Array into which the values of the argument are stored and returned to the calling code.

F: Array into which the values of the function are stored and returned to the calling code.

NUM: Number of data pairs read. Returned to the calling code.

Operation

Subroutine REALIN is used to read a single pair of values of argument and function. If the value of the argument is 99 and the function 0, the sequence is terminated. (Note a zero function value will result if the second data value is omitted by the operator so that the list is terminated by entering 99 on the keyboard.)

The maximum number of data pairs, is set within DATIN at 150, to correspond to the storage allocated for input functions in NERF.

SUBROUTINE FCNFIL

Function

The subroutine FCNFIL provides an interactive facility for the preparation and editing of function files.

Operation

The operation of FCNFIL centres around the code starting with statement number 80 which prompts the operator to select one of the available options.

Prior to reaching this statement, the data for the function file is initialised by either reading an existing file or entering a suitable set of data from the remote terminal. The options and the facilities they then provide are described below.

(1) INSERT

A continuous block of ordered pairs can be inserted immediately following a pair which is identified by specifying the value of the argument.

(11) DELETE

A continuous block of ordered pairs starting with a pair identified by the argument value can be deleted.

(iii) CHANGE

A data pair, identified by the argument value can be changed.

(iv) TITLE

Selected lines of the title can be re-inserted.

(v) CHECK

The contents of the function file can be checked by having them displayed on the remote terminal.

(vi) SAVE

The file can be saved on disk and the operator then has the option of returning to the main program or continuing the editing process for the same set of data.

(vii) CANCEL

The option 'cancel' allows the operator to abort an editing session and permits immediate return to the main program. The operator is given a second chance to confirm this choice as a return to the main program will destroy the current set of data.

The code in FCNFIL is a direct implementation of these actions and should present no difficulties to the FORTRAN programmer.

REFERENCES

1. Payne, A. O. and Grandage, J. M. A probabilistic approach to structural design. Proceedings of the First International Conference on Applications of Statistics and Probability to Soil and Structural Engineering, Hong Kong, Sept. 13-16, pp. 36-74 Hong Kong University Press, 1971.
2. Payne, A. O. A reliability approach to the fatigue of structures. ASTM STP 511 pp. 106-155, 1972.
3. Diamond, P. and Payne, A. O. Reliability analysis applied to structural tests. Proceedings of Symposium on Advanced Approaches to Fatigue Evaluation, ICAF Miami Beach. NASA SP-309, 275-332, 1972.
4. Payne, A. O. and Graham, A. D. Reliability analysis for optimum design. Engineering Fracture Mechanics, v. 12, pp. 329-346, 1979.
5. Mallinson, G. D. On the Genesis of Reliability Models Department of Defence Support, ARL Structures Report.
6. Ford, D. G. Reliability and structural fatigue in one-crack models. Department of Defence, ARL Structures Report 369, 1978.
7. Ford, D. G. Structural fatigue in one-crack models with arbitrary inspection. Department of Defence, ARL Structures Report 377, 1979.
8. Ford, D. G. Coarsely random cracking in one-crack fatigue models. Department of Defence, ARL Structures Report 382, 1980.
9. Hooke, F. H. A comparison of reliability and conventional estimation of safe fatigue life and safe inspection intervals. ICAF, Miami Beach, NASA SP-309, pp. 667-680, 1972.
10. Hooke, F. H. Probabilistic design and structural fatigue. The Aeronautical Journal, pp. 267, 1975.
11. Hooke, F. H. Aircraft structural reliability and risk theory - a review. Proceedings Symposium on Aircraft Structural Fatigue, Department of Defence, ARL Structures Report 363, and Materials Report 104, pp. 299-333, 1977
12. Hooke, F. H. A new look at structural reliability and risk theory. AIAA Journal, v. 17, No.9, pp. 980-987, 1979.
13. Mallinson, G. D. and Graham, A. D. NERF - a computer program for the Numerical Evaluation of Reliability Functions: User Manual. (to be published).

2.

14. Rice, J. A Metalgorithm for Adaptive Quadrature Journal of the Association for Computing Machinery, Vol. 22, No.1, January 1975, pp. 61-82.
15. Fritsch, Kahaner and Lyness. Double Integration Using one-Dimensional Adaptive Quadrature Routines : A Software Interface Problem. ACM Transactions on Mathematical Software, vol 7, No.1, March 1981, pages 46-75.
16. McKeeman, W. M. Algorithm 145, adaptive numerical integration by Simpson's rule. Comm. ACM 5, 12 p.604, 1962.
17. Lyness. Notes on the Adaptive Simpson Quadrature Routine. JACM, Vol.15, pp. 483-495.
18. Osborne. Asymptotic Error Formula for Numerical Quadrature. J. Inst. Maths Applies (1974) 13, 219-227
19. Robinson. Adaptive Gaussian Integration. The Australian Computer Journal Vol.3, No.3, Aug. 1971.
20. Lyness and Kaganove. Comments on the Nature of Automatic Quadrature Routines. ACM Transactions on Mathematical Software, Vol.2, No.1, March 1976, pp. 65-81
21. Ahlberg, J. N., Nilson, E. N. and Walsh J. L. The theory of Splines and their Application. Mathematics in Science and Engineering, Vol.38, Academic Press.
22. Ralston, A. & Rabinowitz, P. "A first course in numerical analysis" 2nd ed., 1978. McGraw Hill, N.Y.

NOTATION

a	Lower limit of integration over x .
a or $a(\tilde{t})$	Crack length, function of age.
a_{con}	Constant value for crack length.
a_d	Crack length inspection threshold beyond which an inspection is 'perfect'.
a_i	Crack length at initiation; or, i 'th coefficient defined by equation (4.99) and used to solve the spline equations.
a_f	Crack length corresponding to the fatigue life limit, \tilde{t}_f .
a_0	Crack length at $t = 0$, i.e., initial crack length.
$a_{0,max}$	Maximum initial crack length.
A_j	j 'th weight in n point quadrature sum, equation (4.14).
b	Upper limit of integration over x .
b_i	i 'th coefficient defined by equation (4.97) and used to solve the spline equations.
c	Constant in density functions.
c_i	i 'th coefficient defined by equation (4.98) and used to solve the spline equations.
$C_d(a)$	Crack detection function.
d_1	See equation (4.91).
D_1	Subspace of uncracked structures.
D_2	Subspace of cracked structures.
D_3	Subspace of structures that have passed a failure criterion.
dt	Small interval of time.
e	Minimum value parameter, (see equation 3.32).
e_s	Scaled minimum value parameter.
$E(f)$	Expected time to failure.
$E(R f > t)$	Expected value for strength.
$E(R t)$	Expected value for the failing load.

$f(R)$	See equation (2.11).
$f(t)$	Function of time (arbitrary).
$f(t, n_0)$	Arbitrary function of t and n_0 , see equation (3.1)
$f(x)$	Arbitrary function of x .
f_{n_0}	Limit for n_0 , defined by equation (2.80).
$f_{n_0}(R)$	Limit for n_0 , dependent on the value of R . Defined by equation (3.72).
$f_{n_0}(R_{\min})$	$f_{n_0}(R)$ for the special case of $R = R_{\min}$.
$f_{n_0}^d(R)$	Limit for n_0 , defined by equation (2.92).
$f_R(R, t, n_0)$	n_0 integrand function for the density for strength. (Equation 3.14)
$f_{R, f_2}(R, t, n_0)$	n_0 integrand function for the contribution to the failure density for strength arising from fatigue life exhaustion.
\underline{F}	Random variable representing the time of failure.
$F_x(x, y, z)$	Integrand function for the x integration of a nested integration sequence, equation (4.41).
$F_y(y, z)$	Integrand function for the y integration of a nested integration sequence, equation (4.40).
$F_z(z)$	Integrand function for the z integration of a nested integration sequence, equation (4.39).
$F_0(a_0)$	Integrand function for the a_0 integration of a nested integration sequence, equation (5.155).
$F_\alpha(\alpha)$	Integrand function for the α integration of a nested integration sequence, equation (5.157).
$F_\beta(\beta)$	Integrand function for the β integration of a nested integration sequence, equation (5.158).
$g(R)$	See equation (2.12).
$g_R(t, n_0)$	n_0 integrand function for the failure density for strength.
$G(\alpha, \beta, n_0)$	Function used in the evaluation of the loss factor, defined by equation (5.60).

$G^*(\alpha, \beta)$	Function used for the evaluation of the loss factor, defined by equation (5.63).
h	Length of integration interval, i.e. $h = b - a$.
h_1	Length of the i 'th interval; used for spline interpolation. (Equation (4.70))
$H(\underline{x}, t)$	Loss factor, equation (2.40).
$H(\alpha, \beta, n_0, t)$	Loss factor, in terms of the random variables used in NERF. Equations (2.43) to (2.45).
$H_S(x)$	Heaviside step function.
i	Suffix, usually a particular value in a series.
$I(a, b)$	Integral of $f(x)$ over the interval $[a, b]$, equation (4.1).
$I_1(x_1, x_{i+1})$	Integration of $f(x)$ over the i 'th interval, $[x_1, x_{i+1}]$.
$I_n(a, b)$	Numerical estimate of $I(a, b)$ using n integrand evaluations.
$I_n(z_1, z_2)$	Numerical estimate of a multiple integration where z is the outermost integration variable.
$I_y(y, z)$	Integrand for the y integration of a multiple integration sequence.
$I_z(z)$	Integrand for the z integration of a multiple integration sequence.
$I_{zn}(z)$	Integrand returned by a numerical evaluation procedure, equation (4.45).
j	Index denoting a particular value in a series. Often used for the age values defining $\psi(\bar{t})$.
k	Index denoting a particular value in a series, or, Dispersion parameter for a density function.
K	Total number of time values for which the reliability functions are evaluated.
K_{lim}	Number of nodes in the α direction for the loss factor interpolation table.
K_1	Number of nodes in the interval $[\alpha_{1j}, R_{max}]$ for the loss factor interpolation table.
K_2	Number of nodes in the interval $[R_{max}, \alpha_{2j}]$ for the loss factor interpolation table.

l_r	Average load application rate.
$l_f(\alpha, \beta, n_0, t)$	Loss factor (see equation 5.183).
L	Random variable representing applied load.
$LG_{j,k}$	Table entry of $\log(G^*(\alpha_k, \beta_j))$, equation (5.129).
LG_j	Interpolated value of $\log(G^*)$ for the j 'th β line.
LG_2	Interpolated value of $\log(G^*)$ for the $j+1$ 'th β line, see equation (5.131).
$m(x)$	Second derivative function as used for spline interpolation, equation (4.69).
m_1	Value of $m(x)$ at $x = x_1$.
M	Number of nodes used to define $f(x)$ for spline interpolation.
n_{d1}	Lower limit for n_0 for the probability of detection.
n_{f1}	Lower limit for n_0 for the risk of fatigue life exhaustion, equation (3.85).
n_{f2}	Upper limit for n_0 for the risk of fatigue life exhaustion, equation (3.85).
n_{R1}	Lower limit for n_0 for the density for strength, equation (3.88).
n_{R2}	Upper limit for n_0 for the density for strength, equation (3.88).
n_0	Initial age.
$n_{0,min}$	Lower limit for n_0 after default limiting.
$n_{0,max}$	Upper limit for n_0 after default limiting.
n_1	Lower limit for n_0 for the probability of failure, equation (3.80).
n_2	Upper limit for n_0 for the probability of failure, equation (3.80).
N	Number of integration levels.
$p_{a_0}(a_0)$	Density for initial crack length.

$P_{ff1}(t, n_0)$	n_0 integrand function for the first $r_f(t)$ integration, equation (3.85).
$P_{ff2}(t, n_0)$	n_0 integrand function for the second $r_f(t)$ integration, equation (3.85).
$P_{fs}(t, n_0)$	n_0 integrand function for $r_s(t)$, equation (3.84).
$P_{fv}(t)$	$r_v(t) \cdot \underline{P}_s(t)$, equation (3.83).
$P_F(t)$	Density for the time to failure.
$P_{n_0}(n_0)$	Density for initial age.
$P_R(R F>t)$	Density function for strength.
$P_R(R t)$	Failure density for strength.
$P_X(\underline{x})$	Joint density function for the random variables \underline{X} .
$P_\alpha(\alpha)$	Density for virgin strength.
$P_\beta(\beta)$	Density for age (given t); generally accounts for previous inspections.
$P_\beta(\beta a_0)$	Density for age given a_0 .
$P_{det}(ti_j)$	Probability of detection at the inspection at ti_j .
$P_{det}(t, n_0)$	n_0 integrand function for the probability of detection, equation (3.82).
$P_F(t)$	Probability of failure.
$P_{\underline{F}}(t, n_0)$	n_0 integrand function for the probability of failure.
$P_L(L)$	Cumulative probability function for the applied load.
$\bar{P}_L(R)$	Probability of load exceedence.
$P_s(t)$	Probability of survival.
$P_s^+(t)$	Probability of survival just after an inspection.
$P_s^-(t)$	Probability of survival just before an inspection.
$P_s^J(t-ti_j)$	Probability of survival of a population of replacement structures, equation (2.50).

$Q_n \{f(x):a,b\}$	n point quadrature sum, equation (4.14).
r	Index denoting level of subdivision in the adaptive Simpson integration algorithm.
$r(t)$	Total risk rate.
$r_f(t)$	Risk of fatigue life exhaustion.
$r_{ff1}(t, n_0)$	$p_{ff1}(t, n_0) \cdot P_S(t)$.
$r_{ff2}(t, n_0)$	$p_{ff2}(t, n_0) \cdot P_S(t)$.
$r_k(t)$	Risk rate for the k'th subspace, equation (2.20).
r_{lim}	Limit risk rate for inspection procedure.
$r_{mean}(t)$	Mean risk rate, equation (3.175).
$r_s(t)$	Risk of static fracture by fatigue.
$r_s(t, n_0)$	$p_{fs}(t, n_0) \cdot P_S(t)$.
$r_v(t)$	Virgin risk (risk rate for uncracked structures).
$r_1(\alpha)$	Instantaneous risk for uncracked structures in D_1 .
$r_2(\alpha, \beta)$	Instantaneous risk for cracked structures in D_2 .
R	Random variable representing strength.
$\bar{R}(a(\bar{t}))$	Median strength decay function.
R^*	$\alpha^* \psi(\beta)$, equation (2.84).
$R_f^*(n_0)$	See equation (2.90).
$R^*(n_0, \beta)$	$\alpha^*(n_0, \beta) \psi(\beta)$, equation (3.54). Lower bound for R given n_0 and β for the D_2 subspace.
R_{max}	Maximum value for applied load.
R_{min}	Minimum value for strength, following default limiting.
R_J	Mesh variable for the β direction of the loss factor interpolation table, equation (5.83).
R_K	Mesh variable for the α direction of the loss factor interpolation table, equation (5.88).
R_0	Median virgin strength.
R_p	Proof load.

$s(x)$	First derivative function; used for spline interpolation, equation (4.64).
s_1	Value of $s(x)$ at $x=x_1$.
$S(\alpha, \beta, n_0, t, t_{1j})$	Inspection removal function, equation (2.52).
$S_j(t)$	Inspection removal function derived from the Crack detection function, equation (5.25).
$S_3^r(x, \Delta x_r)$	Simpson 3 point rule for the r 'th subdivision, equation (4.6).
$S_7^r(x, \Delta x_r)$	Simpson 7 point rule for the r 'th subdivision, equation (4.7).
t	Time.
\bar{t}	Median time, or age.
\bar{t}_d	Age corresponding to a_d (after default limiting).
t_f	Fatigue life of a given structure.
\bar{t}_f	Median fatigue life, after default limiting.
\bar{t}_f^*	Upper bound for β for a D_2 domain bounded below by $R_{\min} = \alpha\psi(\beta)$. $\bar{t}_f^* = \bar{t}_f^*(R_{\min})$.
$\bar{t}_f^*(R)$	Upper bound for β for a D_2 domain bounded below by $R = \alpha\psi(\beta)$, equation (3.50).
t_i	Crack initiation time for a given structure.
\bar{t}_i	Median initiation time (or initiation age).
t_{1j}	Time of j 'th inspection.
t_k	k 'th time value for the evaluation of the reliability functions.
t_n	Specified evaluation time (there are N values).
t_{Rn}	Specified time for the calculation of strength functions.
\bar{t}_0	Initial age (same as n_0).
$T(t)$	General reliability function, equation (5.154).
$T_{n_0}(n_0)$	Term depending on n_0 .
T_2^1	Trapezoidal 2 point rule, equation (4.26).
T_3^1	Trapezoidal 3 point rule, equation (4.27).

$T_v(t, n_0)$	Term for virgin structure contribution to the probability of detection, equation (5.174).
$T_v(t)$	Correction term representing virgin structure contribution to $P_s(t)$, equation (5.180).
$T(\alpha)$	Term evaluated by the function FALP.
$T(\beta)$	Term evaluated by the function FBET.
v	Location parameter, equation (5.33).
v_s	Scaled location parameter, equation (5.36).
\underline{x}	Vector of sample values of the random variables \underline{X} .
x	General coordinate.
x_1	1'th value of x . Used to denote nodes in the spline interpolation or used to define the initial subdivisions in the adaptive integration algorithms.
$x_{1,max}$	Maximum value in the sequence of values of x_1 .
$x_{1,min}$	Minimum value in the sequence of values of x_1 .
x_1	Sample value for \underline{X}_1 .
x_2	Sample value for \underline{X}_2 .
x_3	Sample value for \underline{X}_3 .
\underline{X}_1	Random variable representing relative fatigue life.
\underline{X}_2	Random variable representing initial crack length.
\underline{X}_3	Random variable representing relative residual strength.
y	General coordinate.
$y(x)$	General function of x , esp. spline interpolation.
$y'(x)$	dy/dx .
y_1^D	First order approximation to dy/dx at $x = x_1$.
y_1	Value of y at $x = x_1$.
$y_1(z)$	Lower limit for a y integration in a nested sequence.
$y_2(z)$	Upper limit for y integration in a nested sequence.

\underline{y}	Transformed random variable used to generalise the density functions, equation (5.35).
z	General coordinate.
z_1	Lower limit for an integration over z .
z_2	Upper limit for an integration over z .
\underline{z}	Standard random variable, equation (5.33).
α	Virgin strength (Random variable and sample value).
α^*	Lower limit for α resulting from a proof load inspection, equation (2.58).
$\alpha^*(n_0, \beta)$	Lower bound for α , given n_0 and β for the D_2 subspace, equation (3.52).
α_f^*	See equation (2.70).
α_{high}	Defined by equation (5.87).
α_{low}	Defined by equation (5.106).
$\alpha_{K_{lim}}$	α value on the line $\alpha\psi(\beta) = R_{max}$, see Figure 5.14.
$\alpha_{K_{lim}-1}$	α value on the mesh line next to the line, $R_{max} = \alpha\psi(\beta)$, see figure 5.14.
α_{max}	Upper limit for α .
α'_{max}	Maximum value of the α_{ij} 's, equation (5.79).
α_{min}	Lower limit for α .
α'_{min}	Minimum value for all the α_{ij} 's, equation (5.78).
α_v	Lower bound for α for the D_1 subspace of uncracked structures, equation (3.57).
α_1	Lower limit for the first integration over α made by the function FBET.
α_2	Upper limit for the integrations over α made by the function FBET.
α_3	Lower limit for α for the 2nd integration over α made by the function FBET.

α_{1j}	Lower limit for α on the line $\beta = \beta_j$. Defined by equation (5.74) and used for the loss factor interpolation table.
α_{2j}	Upper limit for α on the line $\beta = \beta_j$. Defined by equation (5.75).
β	Random variable representing age, equation (2.28).
β^*	$\gamma^{-1}(R_{\min}/\alpha)$, as used in equation (2.72).
β'	See equation (3.65). Also used in equation (5.136).
$\beta_d(n_0)$	Limit function for β , defined by equation (3.51).
$\beta_1(n_0, \beta)$	β value defining the translation of the proof load boundary, equation (3.53).
β_j	j 'th value of β , as used to define the function $\gamma(\beta)$.
$\beta_{K \lim}^*$	β value arising during the interpolation of the loss factor near the line $R_{\max} = \alpha \gamma(\beta)$, equation (5.137).
β_{\max}	Maximum value of t/x_1 or $(\beta - n_0)$.
β_{\min}	Minimum value of t/x_1 or $(\beta - n_0)$.
$\beta_{p,R}$	Lower limit for β for an integration along the line $R = \alpha \gamma(\beta)$ in D_2 , equation (3.59).
$\beta'_{p,R}$	Solution of $R = R_p \gamma(\beta) / \gamma(\beta_1(n_0, \beta))$.
$\beta_{p,R \min}$	As for $\beta_{p,R}$ but with $R = R_{\min}$.
$\beta_1(n_0)$	Lower bound for β given n_0 , equation (3.58).
β_1	Lower β limit for integration of $F_\beta(\beta)$, equation (3.156).
β_2	Upper β limit for integration of $F_\beta(\beta)$, equation (3.156).
γ_1	1'th constant, defined by equation (4.93) and used for the solution of the spline equations.
γ_k	Coefficients in expansion for truncation error associated with quadrature sum, equation (4.15).
$\delta(\tilde{t}_f, \tilde{t}_f^*)$	$= 1$ if $\tilde{t}_f = \tilde{t}_f^*$; $= 0$ otherwise.
ϵ	Convergence criterion, typically 10^{-4} .
ϵ_{abs}	Absolute error criterion, equation (4.2).

ε_{act}	Actual error incurred during a numerical integration, equation (4.43).
ε_{est}	Estimated absolute error, equation (4.22).
ε_1	Error associated with integration over the interval, $[x_1, x_{1+1}]$, equation (4.24).
ε_{in}	Numerical parameter used in integration algorithm to control errors, equation (4.44).
ε_n	Error associated with the integration rule, equation (4.46).
$\varepsilon_{n,x}(xz)$	Error associated with the x integration in a nested integration sequence, equation (4.61).
$\varepsilon_{n,y}(z)$	Error associated with the y integration in a nested integration sequence, equation (4.61).
ε_{rel}	Relative error criterion.
$\varepsilon_{y,abs}$	Absolute error criterion used for the x integration, (resulting from the behaviour of the y level integration.)
$\varepsilon_z(z)$	Error associated with the evaluation of the z integrand, equation (4.49).
$\varepsilon_{z,abs}$	Absolute error criterion used for the y integration, (resulting from the behaviour of the z level integration.)
ε'	See equation (4.34).
ξ_j	Coefficients used in quadrature sum, equation (4.14).
λ_1	Coefficients, defined by equation (4.95) and used in the solution of the spline equations.
σ	Standard deviation.
Γ	Gamma function
$\psi(t)$ or $\psi(\beta)$	Relative strength function.
ψ'	$d\psi/d\beta$.
Δx_r	Interval length for the r'th level subdivision in the adaptive Simpson algorithm, equation (4.12).

A.1. Cross Reference Listing for Subroutines and Functions

The table in this appendix lists the subroutines and function subprograms in NERF, NERPRE and NERPLT in alphabetical order. Against each, the page number for the documentation in this report is given together with a statement of the function of that sub-program.

The source code for NERF resides in five files, with the sub-programs arranged in hierarchical order. These files are named NERFPO.FOR to NERFP4.FOR with NERFPO.FOR containing the sub-programs that are highest in the hierarchy. The program NERPLT is stored in a file called NERPLT.FOR and NERPRE in NERPRE.FOR.

The location of each subroutine in the source files is indicated by the numbers 0, 1, 2, 3, or 4 for the NERF routines; PRE for NERPRE and PLT for NERPLT.

Name	File	Page	Operation
ADAPTO	3	195	Outermost integration.
ADAPT1	3	198	Middle integration.
ADAPT2	3	168	Inner-most integration.
ADASET	3	177	Initialise all integration algs.
ADVNC	0	262	Construct time sequence.
ALOG1	2	495	Log function.
ALPHAP	2	434	Find α_p .
ALPSET	2	331	Initialise $p_\alpha(\alpha)$.
ALPTAB	4	356	Find α corresponding to HK.
ARROUT	4	486	Write 2.d. array.
ARRPLT	2	520	Contour map 2.d. array.
BETALM	1	440	Find β limits.
BETCNG	2	332	Change $p_\beta(\beta)$ for new t.
BETSET	2	333	Initialise $p_\beta(\beta)$.
CHANGE	PRE	567	Edit data for control file.
CHECK	PRE	569	Display control file data.
CFIN	0	246	Read control file.
CFNEW	0	286	Update control file.
COMPRS	PLT	551	Window a line.
CONT	4	524	Contour map routine.
CRKDEV	2	307	Evaluate da/dt
CRKGR	2	308	Evaluate $a(t)$.
CRKINV	2	310	Evaluate $a^{-1}(a)$.
DATIN	PRE	574	Input function file from VDU.
DERIV	3	222	Derivative from ordered pairs.
DIAG	4	526	Diagonal interpolation.
ECHO	4	475	Record user responses.
ERRROUT	3	179	Integration error handling
EXP1	2	496	Exponential function.
EXTIME	3	492	Run time monitor.
FALP	1	410	Eq. (5.182).
FBET	1	413	Eq. (5.186).
FCNFIL	PRE	575	Edit function files.
FDSTO	1	418	$P_{det}(t, a_0)$.
FDSR1	1	460	Integrand for $f_R(R, t, a_0)$.

Name	File	Page	Operation
FINISH	3	493	Terminate program operation.
FINTRP	3	219	Spline interpolation.
FLDFO	1	470	$p_{a_0}(a_0)f_{R,f2}(R,t,n_0)$.
FLDRO	1	461	$p_{a_0}(a_0)f_R(R,t,n_0)$.
FLPROB	0	286	Strength functions.
FNOR	1	436	$f_{n_0}(R)$.
FPDET	0	420	Probability of detection.
FPLOTS	PLT	552	Set input functions for plotting.
FPOINT	PLT	554	Plot ordered pairs.
FRFO	1	449	n_0 integrand for $r_f(t)$.
FRF1	1	452	Integrand for $p_{ff1}(t,n_0)$.
FRF2	1	453	Integrand for $p_{ff2}(t,n_0)$.
FRLTO	1	421	General n_0 integrand.
FRP	2	437	$R\gamma(\beta)/\psi(\beta_1(\beta,n_0))$.
FRVO	1	424	$r_v(t)$.
FSIZE	4	533	Character string for a number.
FSOLVE	3	227	Solution of equation.
FSOLV2	3	227	Solution of equation.
FUNC	PLT	555	Input function evaluation.
GADJST	1	383	Discontinuity in loss factor.
GALP	1	376	α -interpolation for loss factor.
GRID	4	523	Draw interpolation grid.
GSTAR	1	378	Evaluate $G^*(\alpha,\beta)$.
GVAL	1	349	Evaluate $G(\alpha,\beta,n_0)$.
HEAD	0	295	Write heading on output files.
INDHI	3	232	Index location.
INDLOW	3	233	Index location.
INFINT	3	199	Summary for integration.
INFLE1	3	201	Error estimate for ADAPT1.
INFLE2	3	202	Error estimate for ADAPT2.
INFST	3	203	Initialise error processing.
INFSUP	3	204	Suppress integration information.
INITAB	4	358	Initialise loss factor interpolation.
INTEST	4	476	Logical input from terminal.

Name	File	Page	Operation
INTGIN	4	477	Integer input from keyboard.
INTPLT	2	512	Contour map of integrand.
IRROUT	4	485	Write 2.d. integer array.
MERGE	3	235	Merge two sequences of numbers.
NERF	0	241	Main program for NERF.
NERPRE	PRE	563	Main program for NERPRE.
NERPLT	PLT	549	Main program for NERPLT.
NODES	1	364	Set nodes for loss factor terms.
OUTPUT	0	296	Write output to files.
P	4	527	Map from array to physical space.
PALPHA	2	334	$p_{\alpha}(\alpha)$.
PAUSCT	PRE	569	Pause while displaying data.
PBETA	2	335	$p_{\beta}(\beta)$.
PDF	2	336	Density function evaluation.
PDFCNG	PRE	570	Edit density function parameters.
PDFOUT	PRE	571	Display density parameters.
PDFSET	2	338	Initialise densities.
PLOTD	4	535	Plot routine for SMOOTH.
PLTPNT	2	515	Plot evaluation points.
PLTS	PLT	556	Plot sequence of ordered pairs.
PLTSET	2	503	Initialise outer integrand store.
PLTSTR	2	504	Store outer integrand evaluation.
PRNO	2	341	$p_{a_0}(a_0)$.
PROMPT	4	478	Message to terminal.
PSI	2	312	$\psi(\beta)$.
PSISET	2	318	Change nodes for $\psi(\beta)$.
PSIDEV	2	316	$d\psi/d\beta$.
PSINV	2	317	$\psi^{-1}(\psi)$.
RANGE	3	229	Range limiting.
READFN	4	448	Read function file.
READMF	PLT	550	Read function file - many functions.
REALIN	4	480	Real number input from terminal.
RFSET	0	251	Set nodes for $p_{ff1}(t, n_0)$
RINTV	1	366	Integral term for loss factor.
RKTAB	4	360	RK from α ; loss factor table.

Name	File	Page	Operation
RLGAM	2	367	Integrand for loss factor term.
RLINE	2	522	Construct $R_{\text{ray}}(\beta)$ on graph.
RLOAD	2	321	$r_2(R)$.
RLOSET	2	322	Change nodes for $r_2(R)$.
RNONRM	2	342	Normalise density for a_0 .
RNOSET	2	343	Initialise density for a_0 .
RSKLOG	1	279	$\log(r(t))$ and other functions.
RSKTOT	1	274	$r(t)$ and other functions.
SCLFCT	4	541	Find plot scale factors.
SELECT	PRE	564	Select new option.
SETGRF	4	536	Initialise plot space.
SETTAB	0	253	Initialise loss factor table.
SINSP	1	325	Inspection removal function.
SMOOTH	4	542	Graphical smoothing.
STRFN	2	319	$\tilde{R}(a)$.
TXIN	4	483	Text input from terminal.
VALOUT	PRE	572	Display data.
WINDOW	PLT	557	Control windowing operations.

A.2. Definitions for variables in COMMON

The variables stored in the various common blocks used by NERF are listed in this Appendix. Each variable is listed with its mathematical symbol, where appropriate and a statement regarding the significance of that variable.

Note that the size of an array variable is indicated and that logical variables are denoted by the word 'logical' in the symbol column. For a logical variable, the condition represented by the value, 'true' is given under 'significance'.

Name	Common Block	Symbol	Significance
ALP1(150)	ALPCOM	α_{1j}	Lower limit for α for the j'th column of the loss factor table.
ALP1IN(100)	INTCOM	α_{1j}	Lower limit for α for the j'th column of the table of integrand values.
ALP2(150)	ALPCOM	α_{2j}	Upper limit for α for the j'th column of the loss factor table.
ALP2IN(100)	INTCOM	α_{2j}	Upper limit for α for the j'th column of the table of integrand values.
ALPC1	CFCOM	k	Dispersion parameter for X_3 .
ALPC2	CFCOM	e	Minimum value parameter for X_3 .
ALPC3	CFCOM	v	Location parameter for X_3 .
ALPCON	INTCON	logical	Variations in X_3 ignored.
ALPMAX	PARCOM	α_{\max}	Maximum value for α .
ALPMIN	PARCOM	α_{\min}	Minimum value for α .
ALPV	PARCOM	α	Current value of α .
AVGLT	RISKCM	$E(\underline{F})$	Expected time to failure.
AVGRS	RISKCM	$E(\underline{F}).P_{\underline{F}}(t)$	
BATCH	BATCH	logical	Current run is in batch mode.
BETA(300)	PSICOM	β_j	Values of β used to define $\psi(\beta)$. Also used to store interpolation tables.
BETC1	CFCOM	k	Dispersion parameter for X_1 .
BETC2	CFCOM	e	Minimum value parameter for X_1 .
BETC3	CFCOM	v	Location parameter for X_1 .
BETCON	INTCON	Logical	Variations in X_1 ignored.
BETINT(100)	INTCOM	β_j	Values of β used to define integrand table, (used for integrand maps.)
BETJD	PSIDIF	β_{jd}	Value of β_j delineating linear or polynomial interpolation for $\psi(\beta)$.
BETMAX	PARCOM	$\beta_{\max} - n_0$	Maximum value for t/x_1 .
BETMIN	PARCOM	$\beta_{\min} - n_0$	Minimum value for t/x_1 .
BETV	PARCOM	β	Current value of β .
BRHIGH	FLDCOM	$\psi^{-1}(R/\alpha_{\max})$	Upper β limit for given R line.
BRLOW	FLDCOM	$\psi^{-1}(R/\alpha_{\min})$	Lower β limit for given R line.
C4D3	ADACOM	4/3	Constant used by adaptive integration routines.

Name	Common Block	Symbol	Significance
CNO	CRLENS	a_0	Current value of a_0 .
CNAME	FNAMES		5 character string defining file for $a(\bar{t})$.
CND	CRLENS	a_d	Inspection limiting crack length
CONT1	FLDCOM		$p_\theta(\beta)/\mu/ \psi'(\beta) $.
CONT2	FLDCOM		$p_\lambda(\lambda)/\psi(\bar{t}_f)$
CONTI	LOGCOM		Not used.
CONTON	INTCON	logical	Integrand maps are possible.
CRK(100)	CRKCOM	a_1	Values of a used to define $a(\bar{t})$.
CRK1	CRKFN		Dispersion parameter for X_2 .
CRK2	CRKFN		Minimum value parameter for X_2 .
CRK3	CRKFN		Location parameter for X_2 .
CURRO	TESTIN	logical	ADAPTO is currently active.
CURR1	TESTIN	logical	ADAPT1 is currently active.
DGS	GDMSWT	logical	The contour map routine, CONT, will interpolate along mesh diagonals.
DINCH	GDMDOT		Length of interval used by the smoothing algorithm.
E(30)	ADACOM	$15/2^{j-2}$	For use in applying (4.13).
EPS	ADACOM	ϵ_{rel}	Relative error criterion.
EPS	CFCOM	ϵ_{rel}	Relative error criterion (temporary storage).
EPS1	SMALCM	10^{-4}	Convergence limit (e.g. for FSOLVE).
EPS2	SMALCM	10^{-7}	Approx 10 x precision of computer.
EPSINT	PARCOM	ϵ_{rel}	Relative error criterion.
ERRES0	INFOR0		Absolute error estimate made by ADAPTO.
ERRES1	INFOR1		Absolute error estimate made by ADAPT1
ERRES2	INFOR2		Absolute error estimate made by ADAPT2
ERRINO	INFOR0		Integration (by ADAPTO) of error estimates made by ADAPT1.
ERRIN1	INFOR1		Integration (by ADAPT1) or error estimates made by ADAPT2.
FCHR	BATCH		First character transmitted in a prompt message.
FINTGO	TESTIN		5 character string identifying the integrand for ADAPTO
FINTG1	TESTIN		5 character string identifying the integrand for ADAPT1.

Name	Common Block	Symbol	Significance
FULSV	LOGCOM	logical	$P_S(t)$ calculated by evaluating integral expression for $P_F(t)$.
G(2500)	GCOM	$\log(G_{j,k}^*)$	Interpolation table used for evaluating loss factor.
GEXP(2500)	GCOM		Space for temporary storage of functions for 2-D maps.
IGSIZE	GCOM	2500	Size of G and GEXP arrays.
INP	PLTPMS		Number of current graph.
INSLOS	LOGCOM	logical	No replacement after inspections.
INSPTS	LOGCOM	logical	Expressions have to account for a previous inspection.
INTOFF	INTOFF		Flag: if $\neq 1$ PLTPNT will not try to plot evaluation points.
ISWT	ISWT		Flag: if $\neq 0$, a loss factor table is being contoured. If $\neq 0$, an integrand map is being contoured.
IT1	RTIME		Initial value of job-time in milli-seconds.
J0	INFO0		Current level of subdivision in ADAPTO.
J1	INFO1		Current level of subdivision in ADAPT1.
J2	INFO2		Current level of subdivision in ADAPT2.
JDIFF	PSIDIF	J	Value of J corresponding to β_{JD}
KALP	CFCOM		Integer identifying standard density function for X_3 .
KBET	CFCOM		Integer identifying the standard density function for X_1 .
KCRK	CRKFN		Integer identifying the standard density function for X_2 .
KLIM	ALPCOM	K_{lim}	Number of nodes in α direction for loss factor table.
L	RFCOM		Number of nodes in α used for evaluating $p_{ff1}(t, n_0)$.
LEMAX0	INFO0		Maximum level reached by ADAPTO.
LEMAX1	INFO1		Maximum level reached by ADAPT1.
LEMAX2	INFO2		Maximum level reached by ADAPT2.
LIMRSK	LOGCOM	logical	Limit risk inspection.
LOGPLT	LOGPLT	logical	Current plot is made on a log basis. (e.g. logs taken before contours drawn.)

Name	Common Block	Symbol	Significance
M	LOADCM	M	Number of data pairs used to define $\bar{P}_L(R)$.
MAXLEO	INFORO		Maximum allowable subdivision level for ADAPTO.
MAXLE1	INFOR1		Maximum allowable subdivision level for ADAPT1.
MAXLE2	INFOR2		Maximum allowable subdivision level for ADAPT2.
N	PSICOM	N	Number of data pairs used to define $\psi(\xi)$.
NCRK	CRKCOM		Number of data pairs used to define $a(\xi)$.
NDC	DETCOM		Number of data pairs used to define $C_d(a)$.
NIMAX	DISCOM		Number of time values for which parameters have been stored for the evaluation of strength functions.
NIMAX	RLIMTS		Number of inspections that have been made.
NINT	INTCOM		Number of nodes in θ direction used for integrand maps.
NLEV	GMSWT		Current contour level number.
NMAX	RNVALS		Number of evaluation times.
NOUT	INTCON		Integer identifying the outermost integration level.
NPO	ADACMO		Number of initial subdivision nodes for ADAPTO.
NP1	ADACH1		Number of initial subdivision nodes for ADAPT1.
NP2	ADACH2		Number of initial subdivision nodes for ADAPT2.
NTERM	NTERM		Number of terms evaluated by FRFO.
NVALSO	INFORO		Number of integrand evaluations made by ADAPTO.
NVALS1	INFOR1		Number of integrand evaluations made by ADAPT1.
NVALS2	INFOR2		Number of integrand evaluations made by ADAPT2.
NVMAXO	INFORO		Maximum number of integrand evaluations for ADAPTO.
NVMAX1	INFOR1		Maximum number of integrand evaluations for ADAPT1.
NVMAX2	INFOR2		Maximum number of integrand evaluations for ADAPT2.

Name	Common Block	Symbol	Significance
PERI	LOGCOM	logical	Periodic inspection procedure has been selected.
PLD	PLDCOM	R_p	Proof load value.
PLTINT	LOGCOM	logical	Integrand plots are required.
PNAME	FNAMES		5 character string identifying the function file for $\psi(t)$.
POPLOS	LOGCOM	logical	The loss factor term is included in the analysis.
PRDET(151)	RLIMTS	$P_{det}(t_{i,j})$	Probability of detection at the j 'th inspection.
PS(300)	PSICOM	$\psi(\beta_j)$	Values of ψ used to define $\psi(\xi)$. Also used to store interpolation tables.
PSIVAL	PARCOM	$\psi(\beta)$	Current value of $\psi(\beta)$.
PSJD	PSIDIF	$\psi(\beta_{jD})$	ψ value corresponding to β_{jD} .
R(300)	LOADCM	R_1	R values used to define $\bar{P}_1(R)$ (or $r_2(R)$).
R12	ADACOM	1/12	
R3	ADACOM	1/3	
RATEL	PARCOM	l_r	Average load application rate.
RDET	RISKCM	$P_{det}(t_{i,j})$	Probability of detection at the last inspection.
RESTRT	LOGCOM	logical	Restart from last run if possible.
RF	RISKCM	$r_f(t)$	Risk of fatigue life exhaustion.
RFARG(150)	RFCOM	$R_{min}/\psi(\beta_j)$	α Nodes used for evaluation of $P_{ff1}(t, n_0)$.
RFRET	PARCOM	$\bar{t}_f^*(R_{min})$	Used for limit calculations for $r_f(t)$ terms.
RFLT	RISKCM	$r_f(t) \cdot P_s(t)$	
RISK	LOGCOM	logical	Expression for $p_f(t)$ is being evaluated. Used to select options in FBET and FALP.
RLD(300)	LOADCM	$\log(r_2(R))$	Values of $\log(r_2(R))$ and interpolation tables for evaluating $r_2(R)$.
RLV	CFCOM		Maximum level of subdivision to be used by adaptive integration routines.
RLMRK(2)	RLIMTS		Limit risk levels. First value determines first inspection. Second value, all others.
RLOADV	FLDCOM	$r_2(R)$	Current value of $r_2(R)$.
RLT	RISKCM	$P_s(t)$	Probability of survival.
RLTDS(10)	DISCOM	$P_s(t_{Rm})$	Probability of survival at a time value for which strength distributions are required.

Name	Common Block	Symbol	Significance
RLTINS	RISKCM	$P_{det}(t_{i,j})$	Fraction of population removed by all previous inspections.
RMEAN	RISKCM	$r_{mean}(t)$	Mean risk.
RMUO	RMUO	\bar{R}_0	Median virgin strength.
RNO	NVALCM	n_0	Initial age.
RNOCON	INTCON	logical	Variations in initial crack length X_2 , ignored.
RNOLIM	CEKFN		Upper integration limit for n_0 .
RNOWAX	CEKFN	$n_{0,max}$	Default, upper limit for n_0 .
RNOMIN	CEKFN	$n_{0,min}$	Default lower limit for n_0 .
RNAME	FNAMES		5 character string identifying function file for $P_L(R)$.
RNC(100)	CRKCOM		Age values defining $a(\bar{t})$.
RND	NVALCM	\bar{t}_d	Age corresponding to a_d .
RNDIST(10)	DISCOM	t_{Rm}	Values of t for which strength distributions are required.
RNF	NVALCM	\bar{t}_f	Fatigue age limit.
RNI	NVALCM	\bar{t}_i	Median initial age.
RNIM	RISKCM	$t_{i,j}$	Time of last inspection.
RNINS(151)	ELIMTS	$t_{i,j}$	Sequence of inspection times.
RNSOLD	RISKCM	t	Within RSKTOT, last time for which reliability functions were evaluated.
RNSV	PARCOM	t	Current value of time.
RNSVAL(151)	RNVALS	t_k	Total sequence of times for which the reliability functions are required.
ROLDLT	RISKCM	$p_F(t)$	Density function for the time to failure; at last time value.
RS	RISKCM	$r_s(t)$	Risk of static fracture by fatigue.
RSLT	RISKCM	$r_s(t).P_S(t)$	
RSLTDS(10)	DISCOM	$r(t).P_S(t)$	Value of $p_F(t)$ at a time value for which strength functions are required.
RTIME	RTIME		Maximum run time, in minutes.
RTOT	RISKCM	$r(t)$	Total risk rate.
RV	RISKCM	$r_v(t)$	Virgin risk.
RVAL	PARCOM	R	Current value of R .
RVALS	CFCOM		Maximum number of functions evaluations for adaptive integration routines.
SBET(100)	DETCOM	\bar{t}	Values of t used to define $S_j(\bar{t})$.
SCX	SMTHSC		Scale factor for plotting in X direction. (units/inch)

Name	Common Block	Symbol	Significance
SCY	SMTHSC	$S_j(\bar{t})$ 10^{-15}	Scale factor for plotting in Y direction. (units/inch)
SPN(300)	DETCOM		Values of $S_j(\bar{t})$ used to define the inspection removal function.
SMALL	SMALCM		Factor used in computing absolute error criterion from ADAPTO integration.
TESTIO	TESTIN		
TESTI1	TESTIN		Factor used in computing absolute error criterion from ADAPT1 integration.
TITLE(2)	TITLE		10 characters for run title.
TXT(20)	TXTCOM		Space for text strings used to create prompt messages.
TXTMP(7)	TXTMP		Additional space for text strings.
VARNO	TESTIN		5 Character string identifying the integration variable for ADAPTO.
VARN1	TESTIN		5 character string identifying the integration variable for ADAPT1.
VIRGIN	LOGCOM	logical	Virgin risk terms included in calculations.
XO	AREAXY		Lower limit of X for plot space.
XCURO	TESTIN		Current value of the integration variable for ADAPTO.
XCUR1	TESTIN		Current value of the integration variable for ADAPT1.
XINTO	INFORO		(b-a) for ADAPTO.
XINT1	INFOR1		(b-a) for ADAPT1.
XL	AREAXY		Upper limit of X for plot space.
XL(30)	ADACOM		Used to calculate Δx_r by adaptive integration routines.
XP0(100)	ADACMO		Nodal values for initial subdivision by ADAPTO.
XP1(100)	ADACMO		Nodal values for initial subdivision by ADAPT1.
XP2(100)	ADACM2	$1/2^{r-1}$	Nodal values for initial sub-division by ADAPT2.
XVAR(3)	PLTMS		Minimum, maximum and tic mark interval for X.
YO	AREAXY		Lower limit of Y for plot space.

Name	Common Block	Symbol	Significance
YL	AREAXY		Upper limit of Y for plot space.
YMO(100)	ADACMO		Integrand values at midpoints of initial subdivisions for ADAPTO.
YM1(100)	ADACM1		Integrand values at midpoints of initial subdivisions for ADAPT1.
YM2(100)	ADACM2		Integrand values at midpoints of initial subdivisions for ADAPT2.
YPO(100)	ADACMO		Integrand values at initial subdivision boundaries for ADAPTO.
YP1(100)	ADACM1		Integrand values at initial subdivision boundaries for ADAPT1.
YP2(100)	ADACM2		Subdivision values at initial subdivision boundaries for ADAPT2.
YVAR(3)	PLTPMS		Minimum, maximum and tic mark interval for Y.

A.3. Alphabetical Listing of Prompts and Error Messages

The various prompts and error messages are tabulated below in alphabetical order. The name of the sub-program in which the prompt is issued is listed against each prompt.

Note that some of the prompts contain numbers or text strings that are inserted at the time the prompt is created. These are indicated by '\$\$\$' in the prompts listed below.

Prompt	Sub-program
ADAPT\$ USING \$\$\$ FOR \$\$\$ FROM \$\$\$ TO \$\$\$	INFST
ANS = \$\$\$ ERR1 = \$\$\$ NVALS-\$\$\$	INFINT
ANS = \$\$\$ ERR1 = \$\$\$ ERR2 = \$\$\$ NVALS-\$\$\$, \$\$\$	INFINT
ANS = \$\$\$ ERR1 = \$\$\$ ERR2 = \$\$\$ ERR3 = \$\$\$ NVALS-\$\$\$, \$\$\$, \$\$\$	INFINT
BETA OUT OF RANGE IN GSTAR	GSTAR
BETCNG ... RNSV ZERO	BETCNG
BOTH SEQUENCES EMPTY IN MERGE	MERGE
CAUTION EXTRAPOLATION X = \$\$\$	FINTRP
CONVERGENCE FAILURE IN FSOLVE ARG=\$\$\$	FSOLVE
CONVERGENCE FAILURE IN FSOLV2 ARG=\$\$\$	FSOLV2
CRACK LENGTH NORMALISING FACTOR IS ZERO	RNONRM
CURRENT VALUE IS \$\$\$ SMOOTHING INTERVAL	SMOOTH
DATA FILE NOT ACCEPTABLE ... G BEING CALCULATED	SETTAB
DOT CODE	INTPLT
DURING THE CALCULATION OF RISK ARE PLOTS REQUIRED?	ADVNC
ENTER AGAIN?	REALIN
ENTER POINT CODE - 1,2,3,4,5 OR 6	PLTPNT
ENTER 5 VALUES AT A TIME - TERMINATE WITH 0	REALIN

Prompt	Sub-program
ERROR IN ADAPT\$ USING \$\$\$ OVER \$\$\$	
ADAPT\$; \$\$\$; \$\$\$ = \$\$\$	
ADAPT\$; \$\$\$; \$\$\$ = \$\$\$	ERRROUT
ERROR IN GAMMA FUNCTION, IFAIL = \$\$\$	PDFSET
ERROR IN FDSR1 ... PSI IS ZERO	FDSR1
ERROR IN MERGE ... XLOW IS GREATER THAN XHIGH XLOW = \$\$\$ XHIGH = \$\$\$	MERGE
ERROR ---- N(\$\$\$) IS LARGER THAN IRMX(\$\$\$)	CONT
EXP1 ... ARG OUT OF BOUNDS	EXP1
FBET ... PSI IS ZERO FOR BETA = \$\$\$	FBET
FINDING RMS CORRESPONDING TO LIMIT RISK	ADVNC
FUNCTION IS ZERO EVERYWHERE ... CANNOT COPE	RANGE
FUNCTION LIMITS LEVELS?	ARRPLT
USE LOG INTERPOLATION?	ARRPLT
G ARRAY READ FROM DISK FILE	SETTAB
GSTAR ... \$\$\$, \$\$\$. IS ILLEGAL	GSTAR
INCLUDE R LINES?	INTPLT
INCLUDE GRID?	GRID
INDICATE MAX LEVELS FOR PLOTTING FUNCTION EVALUATION POINTS BETA ALPHA	PLTPNT
INFORMATION ----- NS = \$\$\$	RSKTOT, FLPROB
INITIALISE INTEGRAND PLOTS BETA VALUES BETWEEN \$\$\$ AND \$\$\$ BETAMIN, BETAMAX, BETASTEP ALPHA VALUES BETWEEN \$\$\$ AND \$\$\$ ALPMIN, ALPMAX, ALPSTEP	SETPLT
INSUFFICIENT DATA TO PLOT	SETPLT
INTEGRAND PLOT?	PLTOUT
INTEGRAND INFORMATION PLOTTED	INTPLT
LENGTH OF X AXIS (INCHES)	ADVNC
LENGTH OF Y AXIS (INCHES)	SETGRF
LIST TERMINATED	SETGRF
LIMITS AND NUMBER OF INTERVALS	REALIN
LIMIT EXCEEDED	REALIN

Prompt	Sub-Program
NEW VALUE FOR NTYP	INTPLT
NO TIME VALUES	ADVNC
NODAL STORAGE LIMIT OF \$\$\$ EXCEEDED	ERROUT
NOT ENOUGH STORAGE FOR CRACK GROWTH FUNCTION	CRGGR
NOTHING TO PLOT	PLTOUT
OUTER INTEGRAND PLOT	
ARGUMENT VALUES ARE BETWEEN \$\$\$ AND \$\$\$	
XMIN, XMAX, XSTEP	PLTOUT
INTEGRAND VALUES ARE BETWEEN \$\$\$ AND \$\$\$	
YMIN, YMAX, YSTEP	PLTOUT
OUTER INTEGRAND PLOT?	PLTSET
PBETA ... ARG IS ZERO	PBETA
PLOT G FUNCTION?	SETTAB
PLOT G INTEGRAL?	SETTAB
PLOT LOSS FACTOR?	SETTAB
PRINT G FUNCTION?	SETTAB
PRINT G INTEGRAL?	SETTAB
PRINT INTEGRAND VALUES	INTPLT
PROBABILITY OF DETECTION	FDET
PROBABILITY OF FAILURE	RSKTOT
PROBABILITY OF SURVIVAL HAS REDUCED TO 0.0	RSKTOT
PROB DIST OF FAILING LOAD CALCULATIONS	FLPROB
PSIDEV ... NO DATA	PSIDEV
R LIMITS BEING CHANGED	SETTAB
R VALUES	INTPLT
RF TERM	RSKTOT
RISK AFTER INSPECTION GREATER THAN LIMIT RISK	ADVNC
RISK AT INSPECTION	ADVNC
RISK FOLLOWING INSPECTION	ADVNC
RISK FUNCTION RSLT	RSKTOT
RUN \$\$\$ TERMINATED	FINISH
STORAGE INSUFFICIENT IN MERGE	MERGE
SUBDIVISION LIMITED X = \$\$\$ LEVEL = \$\$\$	ERROUT
TF IS BEING CHANGED	SETTAB
TI IS BEING CHANGED	SETTAB
TIME LIMIT EXCEEDED	EXTIME

Prompt	Sub-program
USING NTYP = \$\$\$ INTEGRAND MATRIX IS \$\$\$ BY \$\$\$ CHANGE NTYP?	INTPLT
USING NTYP = \$\$\$ INTEGRAND PLOT FAILURE AT J \$\$\$	INTPLT
VIRGIN RISK TERM	FRVO
VIRGIN SURVIVORSHIP TERM	FRVO
** WARNING ** F(HIGH) = \$\$\$	RANGE
** WARNING ** F(LOW) = \$\$\$	RANGE
*** WARNING FOR SURVIVORSHIP CALCULATION *** IF THE RMSVAL INTERVALS ARE LARGE FULL SURVIVORSHIP SHOULD BE USED	ADVNC
** WARNING ** ERROR GREATER THAN REQUESTED	INFINT
** WARNING ** TOTAL ERROR GREATER THAN REQUESTED	INFINT
XMAX (\$\$\$) GREATER THAN LARGEST NODE	ERROUT
XMIN (\$\$\$) LESS THAN SMALLEST NODE	ERROUT
5 VALUES ACCEPTED	REALIN
\$\$\$ ERROR (\$\$\$) GREATER THAN REQUESTED (\$\$\$\$)	ERROUT
\$\$\$ FUNCTION VALUES EXCEEDED	ERROUT

A.4. Program Assembly

The source code for NERF, NERPRE and NERPLT is written in FORTRAN IV as implemented on the DEC System-10 at ARL.

The code for NERF resides in five files;

NERFPO.FOR
NERFP1.FOR
NERFP2.FOR
NERFP3.FOR
NERFP4.FOR

which contain the subroutines in hierarchical order, with the highest level subroutines in NERFPO.FOR. If these files are compiled and searched in the above order they form a library of subroutines which can be loaded into any program making use of any combination of the NERF subroutines without the necessity for multiple passes through the library.

The only routine required by NERF that is not in the above files or part of the standard libraries installed on the DEC System-10 (which include the plot subroutines) is the function that evaluated the gamma function required by PDFSET. The NAG library must be searched to obtain this function.

The program NERF can be assembled by loading the compiled versions of the above files in numerical order and then searching the NAG library.

The program NERPRE resides in the file NERPRE.FOR which

contains the subroutines and functions described in Section 5.11. The program is assembled by loading the compiled version of this file and then searching the main NERF library to complete the assembly.

The program NERPLT resides in a file called NERPLT.FOR and is loaded in the same way as NERPRE.

DISTRIBUTION

AUSTRALIA

Department of Defence

Central Office

- * Chief Defence Scientist)
- Deputy Chief Defence Scientist) (1 copy)
- Superintendent, Science and Technology Programmes)
- Controller, Projects and Analytical Studies)
- Defence Science Representative (U.K.) (Doc Data sheet only)
- Counsellor, Defence Science (U.S.A.) (Doc Data sheet only)
- * Defence Central Library
- * Document Exchange Centre, D.I.S.B. (17 copies)
- Joint Intelligence Organisation
- Librarian H Block, Victoria Barracks, Melbourne
- Director General - Army Development (NSO) (4 copies)
- Defence Industry and Materiel Policy, FAS

Aeronautical Research Laboratories

- * Director
- * Library
- * Superintendent - Structures
- * Divisional File - Structures
- * Authors: G.D. Mallinson
- A.D. Graham

Materials Research Laboratories

Director/Library

Defence Research Centre

Library

RAN Research Laboratory

Library

Navy Office

Navy Scientific Adviser
RAN Aircraft Maintenance and Flight Trials Unit
Directorate of Naval Aircraft Engineering

Army Office

Army Scientific Adviser
Engineering Development Establishment, Library
Royal Military College Library

.../contd.

DISTRIBUTION (CONTD .)

Air Force Office

Air Force Scientific Adviser
Aircraft Research and Development Unit
Scientific Flight Group
Library
Technical Division Library
Director General Aircraft Engineering Air Force
RAAF Academy, Point Cook

Department of Defence Support

Government Aircraft Factories

Manager
Library

Department of Science and Technology

Bureau of Meteorology, Library

Department of Aviation

Library
Flying Operations and Airworthiness Division

Statutory And State Authorities And Industry

Australian Atomic Energy Commission, Director
CSIRO
Materials Science Division, Library
Trans-Australia Airlines, Library
Qantas Airways Limited
SEC of Vic., Herman Research Laboratory, Library
Ansett Airlines of Australia, Library
B.H.P., Melbourne Research Laboratories
Commonwealth Aircraft Corporation, Library
Hawker de Havilland Aust. Pty Ltd, Bankstown, Library
Rolls Royce of Australia Pty Ltd., Mr C.O.A. Bailey

Universities and Colleges

Adelaide	Barr Smith Library
Flinders	Library
Latrobe	Library
Melbourne	Engineering Library
Monash	*Hargrave Library
Newcastle	Library
New England	Library

.../contd.

DISTRIBUTION (CONTD.)

Universities and Colleges (contd.)

Sydney	Engineering Library
N.S.W.	Physical Sciences Library Assoc. Professor R.W. Traill-Nash, Civil Engineering
Queensland	Library
Tasmania	Engineering Library
Western Australia	Library
R.M.I.T.	* Library * Dr A.O. Payne

CANADA

CAARC Coordinator Structures
International Civil Aviation Organization, Library
NRC
Aeronautical & Mechanical Engineering Library
Division of Mechanical Engineering, Director

Universities and Colleges

Toronto Institute for Aerospace Studies

FRANCE

ONERA, Library

GERMANY

Fachinformationszentrum: Energie, Physic, Mathematik GMBH

INDIA

CAARC Coordinator Structures
Defence Ministry, Aero Development Establishment, Library
Hindustan Aeronautics Ltd, Library
National Aeronautical Laboratory, Information Centre

INTERNATIONAL COMMITTEE ON AERONAUTICAL FATIGUE

Per Australian ICAF Representative (25 copies)

ISRAEL

Technion-Israel Institute of Technology
Professor J. Singer

ITALY

Professor Ins. Guiseppe Gabrielli

.../contd.

DISTRIBUTION (CONTD.)

JAPAN

National Research Institute for Metals, Fatigue Testing Division
Institute of Space and Astronautical Science, Library

Universities

Kagawa University Professor H. Ishikawa

NETHERLANDS

National Aerospace Laboratory [NLR], Library

NEW ZEALAND

Defence Scientific Establishment, Library
Transport Ministry, Airworthiness Branch, Library
RNZAF, Vice Consul (Defence Liaison)

Universities

Canterbury Library
Professor D. Stevenson, Mechanical Engineering
Mr J. Stott, Chemical Engineering

SWEDEN

Aeronautical Research Institute, Library
Swedish National Defense Research Institute (FOA)

SWITZERLAND

F+W (Swiss Federal Aircraft Factory)

UNITED KINGDOM

Ministry of Defence, Research, Materials and Collaboration
CAARC, Secretary (NPL)
Royal Aircraft Establishment
Bedford, Library
Farnborough, Dr G. Wood, Materials Department
Commonwealth Air Transport Council Secretariat
National Engineering Laboratory, Library
British Library, Lending Division
CAARC Co-ordinator, Structures
Aircraft Research Association, Library
Fulmer Research Institute Ltd, Research Director
Motor Industry Research Association, Director
Rolls-Royce Ltd
Aero Division Bristol, Library
British Aerospace
Kingston-upon Thames, Library
Hatfield-Chester Division, Library
British Hovercraft Corporation Ltd, Library

.../contd.

DISTRIBUTION (CONTD.)

UNITED KINGDOM (CONTD.)

Universities and Colleges

Bristol	Engineering Library
Cambridge	Library, Engineering Department Whittle Library
London	Professor G.J. Hancock, Aero Engineering
Manchester	Professor, Applied Mathematics
Nottingham	Science Library
Southampton	Library
Liverpool	Fluid Mechanics Division, Dr J.C. Gibbings
Strathclyde	Library
Cranfield Inst. of Technology	Library
Imperial College	Aeronautics Library

UNITED STATES OF AMERICA

NASA Scientific and Technical Information Facility
Applied Mechanics Reviews
The John Crerar Library
Boeing Co.,
 Mr R. Watson
 Mr J.C. McMillan
United Technologies Corporation, Library
Lockheed-California Company
Lockheed Georgia
McDonnell Aircraft Company, Library

Universities and Colleges

Florida	Aero Engineering Department
Johns Hopkins	Professor S. Corrsin, Engineering
Iowa State	Dr G.K. Serovy, Mechanical Engineering
Iowa	Professor R.I. Stephens
Illinois	Professor D.C. Drucker
Princeton	Professor G.L. Mellor, Mechanics
Massachusetts Inst. of Technology	M.I.T. Libraries

SPARES (22 copies) (8 hard copies, 14 fiche copies)

TOTAL (22 hard copies, 168 fiche)

* Hard copies.

Department of Defence

DOCUMENT CONTROL DATA

1. a. AR No AR-002-984	b. Establishment No ARL-STRUC-REPORT-397	2. Document Date September 1983	3. Task No DST 82/011
4. Title NERF - A COMPUTER PROGRAM FOR THE NUMERICAL EVALUATION OF RELIABILITY FUNCTIONS - RELIABILITY MODELLING, NUMERICAL METHODS AND PROGRAM DOCUMENTATION.		5. Security a. document UNCLASSIFIED	6. No Pages 609
		b. title U	c. abstract U
7. No Refs 22		8. Author(s) G.D. MALLINSON A.D. GRAHAM	
9. Downgrading Instructions —		10. Corporate Author and Address Aeronautical Research Laboratories, P.O. Box 4331, MELBOURNE, VIC. 3001	
11. Authority (as appropriate) a. Sponsor b. Security c. Downgrading d. Approval —		12. Secondary Distribution (of this document) Approved for Public Release. Overseas enquirers outside stated limitations should be referred through ASDIS, Defence Information Services Branch, Department of Defence, Campbell Park, CANBERRA ACT 2801	
13. a. This document may be ANNOUNCED in catalogues and awareness services available to ... No limitations.			
13. b. Citation for other purposes (in casual announcement) may be (select) unrestricted (or) as for 13 a.			
14. Descriptors Reliability Mathematical models Fatigue tests Failure Probability density functions Risk analysis		15. COSATI Group 1113 0902	
16. Abstract The computer program NERF (Numerical Evaluation of Reliability Functions) has been designed to evaluate the reliability functions that result from the application of reliability analysis to the fatigue of aircraft structures, in particular those reliability functions derived by Payne ²⁻⁴ and his co-workers at the Aeronautical Research Laboratories. The NERF Program, although based on the Payne reliability models, is capable of extension to more complex models as the need arises. This document details the mathematical development of the reliability functions evaluated by NERF and describes the computer program in sufficient detail to allow desired modifications.			

This page is to be used to record information which is required by the Establishment for its own use but which will not be added to the DISTIS data base unless specifically requested.

16. Abstract (Contd)		
17. Imprint Aeronautical Research Laboratories, Melbourne		
18. Document Series and Number STRUCTURES REPORT 397	19. Cost Code 27 7030	20. Type of Report and Period Covered —
21. Computer Programs Used NERF (FORTRAN) NERPRE (FORTRAN) NERPLT (FORTRAN)		
22. Establishment File Ref(s) —		

ATE
LMED
-8